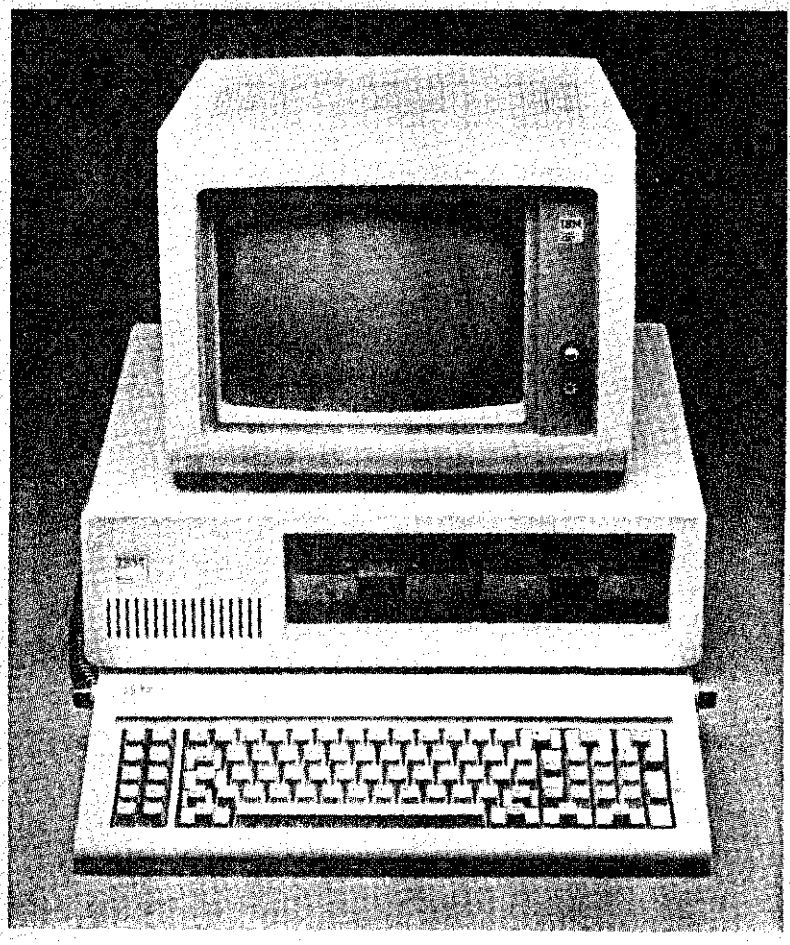
Programmer's Journal the resource journal for IBM PC programmers

EDITORS·ASM·C·REVIEWS



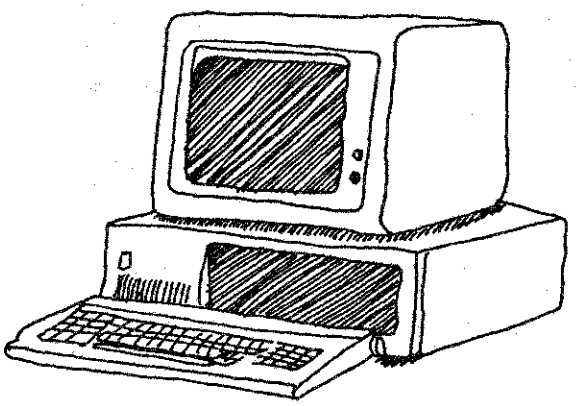
EDITORS·ASM·C·REVIEWS

IBM PC

Office Copy

Volume 1 Number 3 May/June 1983

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



CONTENTS

TEXT EDITORS

IBM EDITORS / Joe Rayhawk 7
PMATE / Greg Estes 12
P-EDIT / Corey Morrell 13
VEDIT / Frankie Detesto 14

ASM: INTRODUCTION TO ASSEMBLY PROGRAMMING / Art Zachai 16
A skeleton program plus a working dumb terminal

PJ PROFILE / Interview with Ken Smolek 20
The author of the C Screen Editor

C SOURCE CODE / Ken Smolek 25
A printer setup utility

PJ FORUM / Dr. Neil Bennett 29
"I'll Be C'ing You"

ASM: GRAPHICS LINE DRAWING PROGRAM / Dan Rollins 32
Fast assembly code for creative graphics

DEPARTMENTS

HOUSEKEEPING 2
LETTERS 3
TOOLS 26
dBASE II 35
UTILITIES 37
PJ LIBRARY 39
SOFTWARE HOUSES 42

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HOUSE KEEPING

HOUSEKEEPING

In March I went to the West Coast Computer Faire. This was the first Computer Faire I had ever attended. For those of you who have not been to one you might ask "Was it worth it the trip?" The answer is a firm yes, at least once a year. This summer there are 2 conventions in San Francisco especially for the PC and others on the East coast as well. The PC '83 Faire on June 17-19 in S.F. should be well worth the trip.

Why was it valuable? I was in the market for a printer and at the convention I got to see 12 different printers all lined up, printing away. Their speeds, print wheels and type faces were all different. Now THAT is the way to pick out a product. 5 of them were on my list of possibilities, but in a few brief minutes only 3 remained. One was thrown out because of cheap construction, and another because of a muddy type style. If you plan on making a major hardware investment then $200-$300 for a Faire trip is money well spent. (Often the Faire prices for hardware are reduced.)

Another jolting experience was standing in front of a COMPLETE programmer's bookstore booth. I thought we had a decent software bookstore here in Oregon, but I was rudely disappointed. There were 100's of computer books I had never seen.

Particular products go well together that you may not get a chance to see otherwise. A Hercules Graphics card on a Monochrome screen in combination with a Mouse is fabulous. You can make intricate line drawings with the sweep of your hand. Architectural drawings, schematics, etc.

If you have never been to a Faire, you should take a long weekend and go. Don't do it for yourself, do it for your MACHINE.

A BRIEF CONFESSION

Not too long ago I had a job working in Wolf Point Montana on the Fort Peck Indian Reservation. This is an extremely northern town on the Canada/Montana border. They called it Wolf Point because of a peculiarity of the area. The winters are long and cold (-40 degrees) and the fur trappers did very well with wolf and beaver pelts. During the trapping season when it was quite cold, you were likely to check your trap run and find a wolf or two. They were, of course, frozen solid and it was impossible to skin them so you just stacked them up next to your shack until the spring thaw. The better trapper you were the higher your pile was. As spring got closer you could travel from one flat town to the next judging a trappers expertise by the point of his stack showing high above the snow. Hence the name Wolf Point.

My work in Montana ended and I needed to buy a car to haul my stuff out West. I picked out a large blue van which was an excellent deal. But after I bought the van I had to face the fact that even though it WAS a good deal, I had rationalized the purchase because I liked the color so much. It was a pale sky blue with a chocolate patina on the horn button. However much I needed the car for pragmatic purposes, in reality I had purchased it for an emotional reason.

I have been through many rationalizations about why I purchased an IBM PC. Of any expensive equipment I own it is undoubtedly the easiest one to justify. I mean you can rationalize it to ANYBODY. You can tell your boss you got one and his eyes light up, obviously you are a talented employee. At lunch your co-workers can eat humble pie when you tell them the tax savings and budget planning potential. Your spouse is easy to convince, try a Christmas mailing list program or a biorhythm chart printout. The kids are a snap, one game and they are in the palm of your hand. Take their game diskette away from them and regain a little power. And for yourself you can use any number of very legitimate reasons. I mean after all we are all programmer's and we can sell this software stuff, we don't NEED any justification.

The confession is this. I did not buy my IBM PC for any of those reasons. The rationalizations finally wore thin and I had to face myself one late night. There are only two reasons why I own a personal computer.

Reason One: I am absolutely addicted to the feeling of learning by doing. Each new discovery on the IBM PC gives me something I can USE right away. What is more self rewarding than that. Mess around and discover a new trick and then implement it in a program or show it to a friend who can use it. It is an extremely personal and solitary experience. I go away for a while and return with some treasure that is of value to me and others.

Reason Two: The advances in science are awesome, and having an IBM PC on my desk is the closest I will ever come to touching the Space Shuttle or owning a piece of tomorrow. Micro computers are incredible, and seeing one every day makes me feel like a participant rather

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

than an observer. Not a moon rock perhaps or an electron microscope, but a piece of the same technology all the same. It is all less foreign to me when I become familiar with my computer.

This time we will look at several editors and some excellent ASM code. Joe Rayhawk gave us an especially good overview of IBM's text editors. Art Zachai and Dan Rollins shared some of the finer points of assembly programming. Neil Bennett, Corey Morrell, and Jim Bumpas passed on their hard won knowledge.

— Greg

Editor/Publisher: Greg Estes
Associate Editor: Frankie Detesto
Contributing Editor: Greg Roberts
Art Director: Julia King
Production: Cassandra Estes
Copy Editing: Renee Anderson

Contributing Writers:
Joe Rayhawk
Arthur Zachai
Neil Bennett
Dan Rollins
Corey Morrell
Jim Bumpas

Editorial and Business Office:
2765 Potter St.
Eugene, OR 97405

Mailing Address:
P.O. Box 5070, Eugene, OR 97405

Programmer's Journal. Volume 1, Number 3.
Copyright (C) 1983 by Programmer's Journal
Published bi-monthly.

Postmaster send address changes to:
Programmer's Journal
2765 Potter St.
Eugene, OR 97405
ph. (503) 687-0398

Subscriptions: Bi-monthly rates are $24 U.S.
($30 in Canada, and $40 elsewhere.)

IBM and Personal Computer are trademarks of International Business Machines, Armonk, New York
dBASE II is a trademark of Ashton-Tate.

CompuServe – 555,670

MY HAT IS OFF

I thought I might write with a few questions and a few comments. First an overall comment on the Journal. On the whole informative though I could personally do without the "testy" aspect of some of your authors and correspondents.

My hat is off to David Whitman for his undertaking and accomplishment. Writing an Assembler, much less in BASIC is really something. I can really sympathize with him in his efforts to learn to program in Assembly language. I do have an IBM Assembler and have been attempting the same. The literature was initially rather sparse but is growing somewhat on the 8086/8088 processor. With respect to books on the subject I find that the "8086/8088 16 bit Microprocessor Primer" by Morgan and Waite from McGraw Hill to be quite comprehensive. In addition there are the manuals by Intel; if you get them, then don't really bother with the "8086/8088 Primer" by Morse since there is a large amount of duplication between it and the Intel books. Clearly acknowledged since Morse was the principal designer of the processor. As noted in Whitman's article, one of the things I have had drummed into me the hard way is that in using the Assembler instructions, you have to look at what is done totally, flags, etc. But once you get started, you can really do a lot with it.

One of my major questions is related to programming in Assembler. On page D-6 of the DOS manual, INTerrupt 27H is described. This is the DOS terminate but stay resident interrupt. It notes at the bottom of the paragraph that this concept (terminate but stay resident) is very useful for loading programs, SUCH AS USER WRITTEN INTERRUPT HANDLERS WHICH MUST STAY RESIDENT. It is followed by a Note warning that this interrupt must not be used by .EXE programs...Thus it is intended for .COM programs. Now to get a .COM file, you have to convert valid .EXE files which have no segment fixup (as stated under EXE2BIN Command) to a form that is compatible with .COM. Now, as I understand it, if you are going to have the program respond to an interrupt, then the interrupt table must have the address of the program. That seems to suggest to me very strongly that the program has to be loaded at an Absolute address so that the table knows where it is in order to call it on being paged. But if it is, you cannot convert it to a .COM file, and according to DOS, you should not use a .EXE file. Seems to me there is some circularity going on.

I noticed another interesting point under the EXE2BIN documentation that there must be no STACK segment

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

along with the statement that it must be in valid .EXE format. The latter requires a STACK and in fact you get an error message if you don't have one. Since DOS does supply a stack, in those cases where I have been successful in converting, the lack of a stack does not seem to hurt. On the other hand, isn't there something illogical about the whole thing?

With respect to DEBUG problems by T. Anderson, I feel somewhat relieved. I have found on more than one occasion when in DEBUG that it totally hangs up and the only recourse I have is to turn off the power. His explanation makes sense. I think that indicates a real need for a hardware reset switch on the IBM PC. Another argument for it, learned from bitter experience, has to do with the fact that as the PC is currently configured, the only way you can reset is via the KEYBOARD. If you are writing programs that utilize KB routines, then you face a real problem in testing them if they have bugs and hangup.

I recently treated myself to 1-2-3. It is absolutely first rate as a spreadsheet, DataBase, and Graphics plotter/printer. Their spreadsheet has a very useful set of Macro procedures available along with IF-THEN-ELSE and GOTO logic. NOT, AND, OR, is included too with a PAUSE command giving you a very extensive interactive programming ability within the spreadsheet. I was able to produce a menu driven Accrual accounting system with it in about one half hour. It is easily learned and the documentation is excellent. Try it, you'll like it. It is quite fast and very flexible. The printer capability that is built into it (you can send all print codes as well as set up strings) is quite extensive. If you use the Macro's, you can produce a very fancy document indeed.

I found the Copy Protection article informative. I had wondered how they did that, particularly since 1-2-3 is protected "benignly." By that I mean that they don't tell you it is protected. I went ahead and copied their System Disk without any problem (I kept wondering why they provided a backup diskette.) Then I went to run the program and I could not get their LOGO to disappear, it just returning me to the startup section. Oh well, whats to worry.

Trust that Programmer's Journal grows healthy.

Morton F. Kaplon
Pomona, NY

EXTERNAL CALLS

I am a fairly experienced BASIC, PASCAL, and FORTRAN programmer. However, in my most recent endeavors I have found a need for producing much faster, more efficient code. Assembly language is/was the obvious answer. But due to the lack of reference material and experience, I found myself frustrated before I even began.

The main problem(s) that I can't seem to find an answer to are these:

1) How does one go about disassembling IBM's BASIC routines for graphics so that they may be used in my own programs? I have several good ideas for some arcade style games that would be perfect for assembly, but the IBM, unlike the Apple, cannot just call monitor subroutines for plotting points or lines or circles. I would like to know how to perform my own graphic operations.

2) What exactly is required by the EXE2BIN program for an assembly language program to be eligible for conversion to a .COM file?

3) How about a means for executing another .COM file or .EXE file from your assembly code.

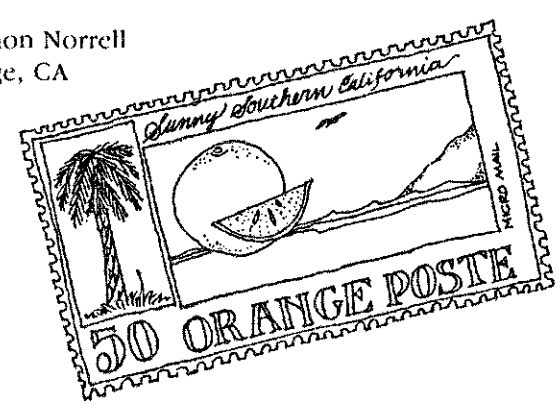
4) How about some more explicit instructions for calling an assembled piece of code from BASIC (i.e. SEGments to be used with appropriate examples.)

These problems, although they might seem elementary to a lot of you more experienced programmers, have become seemingly insolvable for me. I would appreciate it if ANYONE who knows the solution to ANY of these problems would help me out.

Perhaps these problems are common to many other PJ readers and might serve as subjects for future articles.

Sincerely,

Shannon Norrell
Orange, CA



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FLOATING POINT

At a recent IBM PC Club meeting, one person described a problem he was having with a BASIC program. He was adding together amounts of money which totalled about $140,000 but his program was 3 cents off using single precision floating point. When he went to double precision floating point the error was still 1 cent off. The real problem was his accountant; if a pocket calculator could get the right answer, why couldn't the microcomputer (which cost 100 times as much?)

At the time, the suggestion was to use long BCD arithmetic. After thinking about it a bit, a more subtle reason comes to mind. Even in double precision floating point, .1 and .01 CANNOT be expressed exactly. If $1.19 is stored as the number 1.19, then there will be a loss of precision. If the number is stored as 119 cents there will be no loss of precision and his total will be correct. There will need to be a conversion when the final results are displayed but his accountant will be happy.

Regards,

(Dr.) Neil Bennett
Richardson, TX

dBASE II

Just read you very informative article on dBASEII. Thanks for the nice utility.

I have a question for you. Is there any way to override the printing defaults so that I could print 10 pitch instead of 12, or support proportional spacing for special reports or addresses on envelopes whose letters are printed using proportional spacing? I might even (who knows) want to print at 8 LPI instead of six. Is it possible?

Other questions: When, if ever, are they going to upgrade the editor. It is a real pain to have to keep sending files back and forth to my word processor (Benchmark.) It does a WONDERFUL job, I must say. When do you think there might be a color version from Ashton-Tate for those of us who have color?

What I have been doing with my RAM-DISK is only slightly different from what you you mentioned. I often have to deal with some moderately large files (100K or so.) I leave dBASE on drive A and copy my data file onto drive C along with the command files I might need for the particular task at hand; usually using a master file to create a secondary file of labels. This way the program doesn't take up room on the RAM-DISK and I can use 200K for data. The results are a great increase in speed while avoiding wear and tear on my disk drives.

Many thanks for your good work.

Sincerely,

James Narigianis
Brooklyn, NY

(James: Good to hear from you. As far as the printing goes you can just send the necessary ESC codes to your printer and it will do what ever you wish. Check your printer manual for the correct ones. Create a string with the codes you need and then just print them. Set the CONSOLE off and your printer on. Example:

```
STORE CHR(27) + " 8" + CHR(27) + " 6" TO A$
SET PRINT ON
SET CONSOLE OFF
EJECT
? A$
* or ? CHR(27) + " 8" + . . . . . .
SET PRINT OFF
SET CONSOLE ON
```

Be sure and reset your printer after you are finished with the special printing (or turn it off and back on.)

When I am developing a dBASE II program I do not use their editor at all. It wasn't designed to be a knock your socks off editor, nor was EDLIN for that matter. I like to copy everything I need, dBASE, programs, and my text editor over to the RAM-DISK and then bounce back and forth between editing and testing the program with a .BAT file. Works great. Call the .BAT file whatever you wish, (we'll call it BOUNCE) and assuming you are using EDLIN as your editor this is what BOUNCE.BAT would look like:

```
EDLIN MYPROG
DBASE MYPROG
BOUNCE
```

Then when you QUIT out of dBASE II you are back into your editor. A Ctrl-Break at the appropriate point stops it, or you could put a PAUSE command in.

—Frankie

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITORS

TEXT EDITORS

There is a vast difference between a programmer's text editor and a word-processor. A programmer's editor relies heavily on the manipulation of text at the expense of text formatting and fancy printing options. A word-processor on the other hand trys to be a happy medium, a fair amount of text editing with the required pagination, bold facing, and superscripting. If you spend most of your time writing programs and have not used a professional text editor, then you are missing out on one of the finest tools a software writer can own. You may have chosen the editor you have now because your spouse or novice friend would be using it too, and you want to avoid the "computer cramp" that others suffer who do not enjoy these machines as much as we do. The editors described here are unquestionably leaders of the pack. They range in price from $100 to $200. A fast and flexible editor will easily pay for itself in the first month. Once you use one you will never be able to go back to a word-processor except to format a letter or two.

A good text editor will let you manipulate text at any level possible. Move a character, a word, a line, a block, or the whole file. One editor (the Personal Editor) will even let you move a column of text that crosses over several lines. You can also "customize" editors to match your pre-existing keystroke preference. If you use an editor at work and would like to use the same keystrokes at home, no problem.

Every piece of software we use reflects the programmer's ability to translate the following equation:

FLEXIBILITY + POWER = COMPLEXITY

There is a direct connection between these 3 elements, and "user friendly" indicates how well the same amount of power and flexibility can be incorporated with the least amount of complexity. Certainly there is an additional cost in terms of complexity for every feature that is added to an application. Either a COMMAND or a new KEYSTROKE is normally added to operate an enhancement. With WordStar we keystroke ourselves to death, and with operating systems we commandline ourselves into oblivion. The only other interface we have are menus, a nice change once in a while but much too limited in scope. Pick your poison, commands or keystrokes; this is how we handle detailed conversations with computers. Professional text editors are no exception. For most of them however our conversations can overlap into either commands or keystrokes, most of the functions are duplicated in either mode.

Of the BIG FOUR editors we are reviewing, Peter Norton uses VEDIT, Will Fastie puts P-EDIT at the top of his list, the Personal Editor has received rave reviews from several programmers (including Joe Rayhawk), and P-MATE is the editor of choice for many other professionals. All of them are FAST and flexible. They have depth, meaning you cannot hope to learn them completely in half an hour. They all have commands in reserve for when you need to complete a special task (like search 12 source files for an occurrence of "GOSUB WRITE-CUSTMAST-RECORD".) It is refreshing to know that the ultimate tool a programmer will use has been done so well by these companies. You will enjoy them every day you use them. The main-framers who have never used a dedicated micro text editor don't know what they are missing.

—*Greg*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THREE IBM EDITORS FOR THE PC:

EDLIN, Personal Editor, Professional Editor

by Joe Raybauk
Portland, Oregon
ph. 503-626-3131

While reviewing the IBM PC Personal Editor I decided to go ahead and describe its two brothers: EDLIN and the Professional Editor. Although this may seem a little masochistic, it provided me with the opportunity to take a closer look at the Professional Editor. In the process I'll mention a few appropriate ways to use EDLIN.

First, let me give you the bottom line so those of you who are as busy as I am can get back to coding.

The Personal Editor is the *finest* editor I have seen for programmers' use. If you are a semi-skilled typist and reasonably bright (that is a typical programmer), you ought to try the Personal Editor for about 4 hours. You will not go back to whatever you are using now.

The Professional Editor is a very nice extension of the VM XEDIT type editor to the PC environment. It is however like Wordstar in that it is most useful if you use it a lot and you are a good typist.

The EDLIN program can be useful but it has some very nasty faults. If you compensate for these, it is an acceptable editor for small efforts.

INTRODUCTION

What I will try to do is review each product along these guidelines:
- General description
- Does it meets basic functional requirements?
- How good are its human factors?
- Does it have any drawbacks
- Other comments

I will then review the products in ascending order of quality. (If I didn't save the good stuff for last you might not read the parts about EDLIN.)

OK, so what are the basic functional requirements? A decent editor should allow the following:

Create a file from scratch.
Edit an existing file.
List a requested number of lines in the file.
Insert Lines into a file.
Delete Lines from a file
End an Edit Session Saving changes to disk.
End an Edit Session without saving changes.
Locate a character string.
Change one character string into another
Move a block of text.
Copy a block of text.
Allow part or all of another file to be brought into the middle of an active file.
Save a changed file to disk with a new name.

As you come to the end of this list you may be questioning whether the last items are fundamental. I think most programmers will accept the early items. Also, you may have some additional items you would add. I have seen editors with well over 100 commands and distinct options(VM's XEDIT, for example). I have never seen an editor that could do everything I have wanted, nor do I really want one since I probably could not lift its instruction manual. (DOS 2.0's is bad enough.) My general criteria is that a reasonable editor does most of the basic functions without to much hassle. A good editor does them all plus some more neat ones without increasing the keystrokes needed to accomplish the basic functions.

edlin

EDLIN (Part of PC/DOS ($40 for DOS 1.0/1.1, $60 for DOS 2.0)

EDLIN is not a very good editor. It is not even a very good line editor. It does have some important redeeming features that you should bear in mind. Also, the DOS 2.0 version actually provides most of the basic functions.

One redeeming feature of EDLIN is that it is free. (I consider the three purchases of DOS: 1.0, 1.1, and 2.0 to be among the wisest purchases I have ever made.) Since the other functions of DOS are worth much more than the purchase price, whatever I can get out of EDLIN is gratis.

If you are not using your PC for profit, it may be that this is a sufficient criteria for using EDLIN. You should see some of my CAVEATs below however.

Another redeeming feature of EDLIN is that it can be very productive to use for changing one or two lines of a file. This may be needed after running a compiler and getting one or two error messages. (This happens to me about once every two or three years!!) EDLIN is small and it does not clear the screen when invoked. Hence, you can load it quickly; locate the line(s) in error and fix them while still viewing the error message(s). For this cycle, EDLIN can be more effective than using the full-screen programs.

It is debatable whether it is worth learning EDLIN for this purpose if you already have a good full-screen editor.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FUNCTIONAL REQUIREMENTS

The DOS 1.0 and 1.1 version could do most of the basic functions except: MOVEs, COPYs, GETs and SAVE with a new name. The lack of MOVEs and COPYs implies that the author of EDLIN (who hopefully will remain unnamed for his safety) did not use it for his own work. The DOS 2.0 version adds MOVEs, COPYs and GETs to the functions. This moves it into the almost reasonable category.

HUMAN FACTORS

The basic syntax of EDLIN commands is: start line, end line, and command [optional parameters.] I personally find this to be totally backwards because I think in terms of: command, parameters, start line, then end line. The EDLIN approach reminds me of my initial negative reaction to the PDP-11 SOS editor. (I have been told that SOS stands for Son Of Stopgap. The story goes that DEC was about to release their first machine and operating system when someone noticed that they did not have any way to create source files. Enter STOPGAP. Apparently, STOPGAP met with considerable market resistance, hence SOS.)

EDLIN has some nice human factors in that it uses some of the standard DOS calls to read/write terminal buffers. This allows some keying tricks with inserting characters into the middle of existing lines without actively typing the old ones. (The cursor right key causes them to be displayed one by one as though you were typing them in again. The F3 key effectively re-types the rest of the old line.) Also, the fact that the calls use the same code as DOS allows you to learn the same useful tricks for re-entering DOS commands and EDLIN data.

DRAWBACKS

EDLIN has several defects that can be very serious. If you are going to use EDLIN or you are going to have a neophyte use it you should understand these and compensate for them.

First. While EDLIN is editing a file, it leaves the file open as far as DOS is concerned. There are various ramifications to this, but the most important one is that you should not swap disks while you are in EDLIN. If you put in another disk and End an EDLIN session, it will almost certainly destroy useful information on the swapped-in disk. (Basically, DOS does not think you would be foolish enough to remove a disk containing a file that you are actively reading or writing (OPEN). So when it comes time to close it, it assumes that its previous knowledge of where it is ok to write on the disk is accurate.)

Second. when EDLIN goes to write out your file and your diskette is full, it issues a cute little message and exits to DOS. It throws away whatever did not fit. (Some of your data may or may not make it to a temporary file.) The first time this happened I went into shock. (My wife claims I was actually gibbering unintelligibly.) This is also the reason that several Portland IBM PC Club members have offered contracts on the fingers of the EDLIN author. Hence, rule number two is that you do not use EDLIN on a full disk and make a lot of changes, especially additions, to a file. This brings us to Corollary One: If in EDLIN and you think your disk might be full, remember rule number one: Do Not Swap Disks While In EDLIN!

Finally, the Command to Exit EDLIN is E. This command should only work when entered on a line by itself. Unfortunately, it will also work when accidentally entered with the Start Line or End Line Specifications. If you inadvertently enter E as the first character when not in Insert or line edit mode it will obligingly EXIT too.

OTHER COMMENTS

I have taught about 30 people to use EDLIN as part of an Introduction to DOS class. Even total newcomers to computers can pick up most of the concepts in a one hour class, half-lecture-half-hands-on. Hence, it is not an unreasonable introductory tool. After any significant use, they will outgrow it, of course.

professional editor

THE PROFESSIONAL EDITOR ($130)

The Professional Editor is a good full screen editor. It attempts a reasonable balance between a full menu-driven system and a command-driven one.

Digression: a full menu-driven system requires you to go through a series of option menus, selecting the appropriate option to get you to a screen upon which you accomplish your desired function. The advantage of a menu-driven system is that the less familiar user can accomplish his task(s) by reading the screen(s). This can be much more productive than any static user's manual, no matter how well-written. The disadvantage of a menu-driven system is that competent users are impeded by the need to stage through the screens. Most competent users are usually better off with nice simple command syntax. A disadvantage of a command approach is that it can be very hard to remember the particular syntax of an infrequently-used command. Consequently command-driven systems often have Help Menus added that are little more than listings of the commands' syntax.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Professional Editor combines a number of menu-type screens along with a slight variation of its own. Like most full-screen editors it displays the following:

— A window into the data in the file. By moving the cursor into this area, you can directly type over, insert, or delete characters.
— A command area in which you can enter things like locate or change commands.
— A message area through which the editor reports to you.

The Professional Editor adds two more areas. Picking up an idea from the IBM VM editors, it reserves the left columns for Line Commands. These are used to mark blocks of lines for MOVEs, COPYs and DELETEs. Active definitions for the ten Function Keys appear on the bottom of the screen. A cute trick here is that it uses F1 to rotate through four sets of these. This allows you to rotate through four mini-function menus very quickly by hitting F1.

FUNCTIONAL REQUIREMENTS

The Professional Editor supports all of the basic functions with relatively little effort. In some cases, you have to be willing to walk through some menu-screens to get there. I would rate the ease-of-use as very good with only minor jabs in comparison to the Personal Editors.

HUMAN FACTORS

This is a very well-designed program for someone who does a lot of editing work. In particular, it has some features that could be of great assistance to a legal secretary who has to key large amounts of boiler-plate text and phrases. The editor allows a large number of keys to be defined as MACRO keys. For example, pressing Alt-W might cause the phrase: "Whereas the party of the" to be inserted into a legal document. The benefits would accrue only to someone doing a lot of typing with motivation to set up a good set of MACROs.

Given this kind of use and the generally good design, the Professional Editor can be tuned to an individual very nicely.

DRAWBACKS

One Minor One: This editor only supports 80 column displays. Do not try to use it on a low-resolution graphics screen. Several areas are displayed in reverse video and are essentially unreadable on low-res screens. Since some of the unreadable information is extremely vital, it leaves something to be desired.

One Serious One: This editor will only edit 64 Kilobytes at a time. It does play some games while Getting & Putting parts of a larger file. This seems more understandable than with EDLIN, but is still not nice. Also, many easily understood commands change meaning subtly and begin applying only to the current resident segment instead of normally covering the whole file. I judge the subtlety of difference to be dangerous.

The editor is limited to line lengths of 140 or less. This probably comes down from a large system design where the widest line for editing is a standard printer report of 132 characters. At any rate, 140 is kind of a strange number. I think this is a minor weakness in terms of most programming work, few listing are wider than this. However, it is still a weakness because you might very well create a wider report format using compressed mode on an EPSON MX-100 (200+ characters).

OTHER COMMENTS

I have not really attempted to delineate the functions of this editor because they are too numerous. It can do almost everything that the Personal Editor can do, although not always with as few keystrokes or as quick.

I will mention a few that the Personal Editor does not have since you may have a need for them. First, this editor can do right justification and centering of text. Second, the editor can force all input characters to uppercase. Third, you can limit certain commands to act within left and right boundaries. There are many others.

The Professional Editor appears to be a logical outgrowth of several of IBM's best large system editors, notably the VM XEDIT. If you are frequently using the large IBM systems, you may find it easier to switch to the Professional Editor.

Now a real BAD/GOOD point: The Professional Editor and the Personal Editor attempt to do you a favor by saving files to disk and replacing long strings of blanks with TAB characters. This turns out to be very nice for most of your PC program language files. It appears that most of the Microsoft compilers and the Assembler treat TABs as blanks. In almost all cases, this means that your compressed file produces the same output from the compilers as an uncompressed file. The compression seems beneficial. There are some bad aspects though. If you attempt to upload a compressed file to a large system, it may or may not work out as planned. Also, if you edit a data file whose column alignment matters, you might save a compressed version that causes more problems. Both editors allow you to save without compression, but they make it easy to slip up.

At the end of the last section, I will include some BASIC code that could dynamically uncompress a data file while you are reading it.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

personal editor

THE PERSONAL EDITOR ($100.00)

The Personal Editor is a full-screen command-driven editor. It takes remarkable advantage of the IBM PC keyboard. It can be rapidly and easily tailored to the typing needs of a wide range of potential users. It is astoundingly fast at what it does. It is in the same class of product as the spreadsheet 1-2-3. That is, it was written for the IBM PC architecture from the word go.

Having heard rumors about the program for several months before its official announcement, I bought one of the first copies in Portland. I demonstrated it at the Portland Club meeting to a literally awed audience. Within 24 hours, every store in the area was sold out.

I would guess that the majority of IBM PC Programmers will be Personal Editor users.

The editor can handle up to twenty files in memory at once. It can edit files up to the limits of memory. When the memory limit is hit, it will open a spill file on the default disk. This file can hold up to 128 kilobytes. I have used the editor as a memory tester by replicating parts of a file and copying parts around in the active display.

The editor allows 99 keys to be defined to mean sequences of operations. For example: the default definition for the F1 key is:

```
def f1 = [cursor command] [begin line] [erase end line]
         'e pe.hlp' [execute]
```

This means: Move the cursor to the command line, move it to the beginning of that line, erase everything to the end of the line, type the characters: e pe.hlp , then logically press ENTER. If you press F1, you will bring up the Personal Editor's default HELP file for review. You can change these definitions very easily. You can also call up the current active definitions as a file. This can be saved as a new macro type file. Whenever you need this set, you can enter: MACRO Filename on the command line and you are back to the special environment.

BASIC FUNCTIONAL REQUIREMENTS

The Personal Editor can perform all the basic functions easily and quickly. The major COPY and MOVE functions are performed by using the Mark concept. A set of lines of text, a block of characters, or a set of characters can be MARKed. This set can then be copied or moved to elsewhere in the same file or you can make another file active and copy or move into it.

The Personal Editor also does a lot of other desirable functions with few keystrokes and great speed. In particular, it can pop from one full screen to another very quickly. It displays a new full screen faster than any other PC product. It will reformat paragraphs with margins very quickly. Note, it does not do a right-justify of the text. That is, it will leave a ragged right margin.

HUMAN FACTORS

The Personal Editor facilitates the creation and manipulation of program source files by programmers as well or better than any other program I have ever used. I truly doubt that I am going to see a significantly better programmer's editor before voice input is practical. (This includes the Mickey Mouse foolishness announced with the Apple LISA system.)

An interesting point about the Personal Editor is that it is hard for an old editor user to understand just how good the Personal Editor really is. (Many users of line editors were slow to switch to full-screen editors because they looked at them as simple extensions of their current environment which they are not.)

DRAWBACKS

Any caveats regarding the Personal Editor are few and minor. The only one that gets me sometimes is that the only way to avoid tab substitution is to request NOTABS on the command line when you do a FILE or a SAVE. The next version should allow a command like: SET DEFSAVE TABS and SET DEFSAVE NOTABS. This should be executable in a MACRO file.

Several other ones have cropped up with DOS 2.0. In particular, if you are using DOS 2.0 file directories, you can invoke the Personal Editor from any directory. The editor does not know how to find the default key definition file (called PE.PRO) anywhere other than the active directory. So, you come up without the full key definitions. The editor may not allow you to call up other files not in the current active directory. This particular problem is endemic to most of the program development software right now with DOS 2.0. For example, the Linker can be invoked, but it cannot find the Library which is squirreled off in a different File Directory. Hence, these are more a DOS 2.0 than an Editor problem, but it may hang you up a little on using hard disks efficiently.

OTHER COMMENTS

The default key definitions define Ctrl-End to take you to the bottom of the file. Since Ctrl-End does an Erase-to-End of Line in the BASIC interpreter which you cannot change, this definition can cause trouble. I changed it to match the interpreter.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I also re-defined Ctrl-PgUp and Ctrl-PgDn to be Top-of-file and Bottom-of-file. I also extended the definition of a number of keys to force [replace mode] in case I was in [insert mode]. These include ENTER, PgUp, PgDn, Ctrl-PgUp and Ctrl Pg-Dn.

FINAL NOTE - - Code What Else

I have begun using the following BASIC code to adjust records that I am reading which come from files that might have been edited using the Personal Editors. I use as one of my design criteria that data files should be editable with a text editor whenever possible. This allows you to recover from program, user or machine errors more quickly. It also allows you to create test files with real or intentionally garbaged data.

The following code assumes you have just read a line from a file into a variable called: NL$. The code itself might be in an external file called EXPTAB.BAS kept on the B: disk.

For Example:

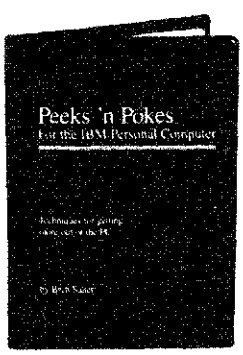
```
1010 LINE INPUT #2,NL$
1020     GOSUB 6600      'Expand TABs, If any.
1030 REM Process Line as appropriate
...
...
6600 REM INCLUDE$: 'B:EXPTAB.BAS'
```

Contents of B:EXPTAB.BAS:

```
6601 REM Subroutine to Expand Embedded Tabs
6605 I6600% = INSTR(NL$,CHR$(9)) 'CHR$(9) = ASCII
     TAB
6610 IF 0 = I6600% GOTO 6625  ' = 0 implies no more
     TABs
6615 NL$ = LEFT$(NL$,I6600%-1) + SPACE$(8-((I6600
     %-1)MOD 8))  + MID$(NL$,I6600% + 1)
6616 ' NL$ = String up to TAB + pad out to multiple of 8
     + string right of TAB
6620 GOTO 6605   'Loop to see if more TABs
6625 RETURN
6699 '
```

Although this involves increased cycles for the BASIC program, it may in fact result in faster execution for two reasons. First, if you do save a file compressed, you or the end user may waste considerable Human and CPU cycles trying to figure what happened. Second, since the compressed file is smaller, it may get processed with fewer DOS level cycles. I have several applications that do run faster because of this. □

Peeks'n Pokes For the IBM PC puts you in control of your PC!



Are you writing programs in BASIC or Pascal? This unique product has a disk with 58 programs and a 38-page manual that helps you get 'underneath the covers' of the PC. Learn how to use PEEK, POKE, INP, OUT, and DOS/BIOS function calls to do what you want, fast! Do you want to perform functions not available from BASIC or Pascal? It's all explained in the manual and demonstrated in the sample programs. You can increase the capabilities of your programs while making them even easier to use.

The easy to learn techniques save you time and increase your productivity.

Peeks 'n Pokes shows you how to:

- Access the system's configuration
- Unprotect BASIC programs
- Scroll part or all of the screen
- Access the file directory
- Logically swap printers
- Read and change the keyboard
- Find more Peeks 'n Pokes
- And much more...

All this for only $30.00 plus $2.50 shipping. MasterCard and VISA accepted.

Data Base Decisions • 14 Bonnie Lane • Atlanta, GA 30328 • 404/256-3860

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

pmate

PMATE

by Greg Estes

PMATE was written by
Phoenix Software Associates Ltd.

It is available from:
Lifeboat Associates
1651 3rd Ave.
New York, NY 10028
(212) 860-0300

PMATE is a superb text editor. It has tremendous depth. Besides all of the normal cursor control keystroke functions it has a COMPLETE set of text handling commands. The macro/command function is so comprehensive that you could almost view your text as a data file with PMATE supplying the programming language to manipulate it. You can manipulate text in every conceivable way possible (by character, word, line, block, and complete file.) This can be done by keystrokes or at the command line.

PMATE has all of the standard functions you expect from a good text editor. We won't spend the time mentioning them here but instead cover it's more advanced functions. Suffice it to say you can move the cursor around the screen with the keypad keys and you can setup the function keys to execute any macro commands you have written. The normal MOVE,INSERT,DELETE is accomplished by the appropriate CONTROL-THIS, CONTROL-THAT keystroke. This can easily be done at the command line too.

PMATE has the command line at the top of the screen. It perpetually shows you the drive you are logged onto, the input file you are using, the output file, and the Line and Column your cursor is at. There are really two cursors. One is the blinking one that floats around the text, and there is another one on the command line. Your text cursor remains in the text at all times, there is no switching back and forth.

CUSTOMIZATION:

This is handled much like the Personal Editor. You have a ASCII file with a bunch of questions in it. The answers you supply determine which keystroke sequences do what. There are numerous other choices, buffer sizes, BAK file creation or not, begin in command mode or overtype mode, and so on. With so many text editors offering customization (PMATE, VEDIT, Personal Editor, etc) it seems almost a prerequisite for an editor to have this feature. An additional option allows you to customize the editor while using it, and then if you happen to like the current setup, you can just create a duplicate version when you exit.

DOCUMENTATION:

This is too often a real complaint when it comes to full blown text editors. Does the manual look like an old IBM OS DOS VS2, ver. 3.2 that has been shuffled twice to maximize confusion (but the information is all there if you could only find it?) Happily I can say that the PMATE manual is very well done. The first part is a novice user's guide to get you going in visual mode, then it moves on to more complex issues. There is a well laid out command summary at the end of the manual.

So much of a manual's friendliness depends on the PROGRESSION of what you are taught. We have all experienced the manuals that tell you how to get INTO the system and then when you want to get OUT you are on your own. Many times I have refused to spend the 1 hour necessary to find the EXIT command in a manual when I am first using a piece of software. I confess I just turn the machine off and assume I will run across the EXIT command when the manual writer finally decides to give it to me. I would use SuperCalc's manual as a model of excellent documentation. The chapters are laid out as you logically need to know the information. You first need to know how to GET IN, how to do a little something, and then how to GET OUT.

PERSONAL PREFERENCES:

We all have particular tastes when it comes to editing features. Since most of the first class editors offer customizing of the keyboard, we can be sure that from the word go 80% of our desires will be met. Whichever control sequences you prefer can be configured with PMATE.

There are 2 features that I hold very dear to my heart. One of them is being able to hit the RETURN key without splitting the line. I have never enjoyed this idiosyncrasy in many editors. PMATE is one of the few editors that allow you to have it both ways. INSERT mode will always split the line, and if you are in OVERTYPE mode you can take your pick; to split or not to split, that is the question. Both VEDIT and Wordstar force a line break when the carriage return is hit.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The other personal taste I try to cater too is having a screen full of text and ONLY text. No ROW COL or other garbage to distract me. PMATE does not to my knowledge offer this. But I acknowledge that we cannot have everything in an editor unless we write it ourself. This may well be a low priority on your list.

PRINTING:

PMATE does not have the automatic print formatting controls that most word-processors have. The standard printing functions for programming editors usually consist of just printing the text to the printer. You can manually insert ESC codes to do any formatting you want. PMATE, with it's extensive macro command ability allows you to write your own text formating macro. It might look like this:

```
[
B2K
B2E
B1G
XT
10QT
BTE
60XT
4[10QT]
@T=0]
```

This is not a complete macro, it is just to give you an idea of the short-hand you must learn to write a series of commands to do a sophisticated printout with page numbering etc.

NICE FEATURES:

You can POP garbage from a stack. There is a junk buffer that fills up with deleted lines. If you delete something you wish to recall you can just pop the last line(s) back into your text. You may setup the size you want this garbage area to be.

PMATE allows LONG lines that scroll horizontally. 250 characters or so. (When it scrolls and you decide to have some fun with this feature it almost looks like you are playing the DEMO game supplied with the BASIC compiler. A pleasant break from mundane editing tasks.)

10 buffer save areas. These can be hold text you wish to have on hand from either your main text, from memory, or from disk. Macros can be loaded and executed from any buffer area.

PMATE is an excellent tool for serious programmers. □

p-edit

PEDIT

by Corey Morrell

PEDIT
Satellite Software International
288 West Center
Orem, Utah 84057

PEDIT, not to be confused with IBM's Personal Editor, is a powerful text editor produced by Satellite Software International of Orem, Utah. The editor itself takes up 16K of memory. It uses an 'edit buffer' to hold the text being edited in memory; this memory buffer will be of varying size, depending on how much memory there is in your system. The maximum memory buffer size, however, is 64Kb. All text exceeding your memory buffer size is stored on two overflow files on disk. These are 'virtual' files, and are therefore still part of the editable text. So, you can see that the size of your text file is only limited by available disk space.

PEDIT is key-driven. This means that all commands are given by depressing different keys, usually in conjunction with the ALT or CTRL keys. Two templates that come with the editor fit onto the IBM keyboard and indicate the keys' functions. One runs above the top row of keys, and the other fits around the function keys on the left. It was somewhat disconcerting at first to use only keys to work the editor, rather than typed in commands, but with a little use it became easier and faster.

One thing about PEDIT that I like better than say, EDLIN, is that it is a 'Full Screen' editor. One can move the cursor anywhere on the page using the four arrow keys, and then insert or overtype characters in place. The arrow keys, used with the HOME key, will also move whole screens up and down, or right and left, if there are lines longer than what will fit on the screen. If I 'scroll' up or down, and the next page is not in the memory buffer, it will be 'paged in' from one of the two virtual files on disk.

PEDIT also makes deleting easy. Using the proper key sequences, I can delete a character, a field or word, a line, a page, or the whole text. Each delete key sequence can be executed several times by preceding it with an 'ESC n', where n is some integer.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If I want to search for a particular string in a text file the search keys will do just that. I can search forward or backward from the cursor for the string I entered. If I omit the string when prompted, PEDIT uses the last string I entered, thus enabling multiple searches for the same string by merely hitting the forward or reverse search key. I can also search and replace. When I hit this key, I am prompted for the search and replace strings. PEDIT will then ask if it should replace every occurrence, or stop at each occurrence for confirmation of the replacement before continuing. Failed searches return the cursor to where it started.

SOME FANCIER STUFF

If I want to move words, lines, or paragraphs around, PEDIT has a range definition key. I can define a range (one character or the whole text), and then move the range elsewhere. I can save the range to a file on diskette, and insert it later into another file during a different PEDIT session. That's a nice feature. This also allows unlimited 'save areas' for cutting and pasting. Once a range is saved in a disk file, I can define other ranges and append them to the existing file, thus piecing together programs from parts of existing programs. For immediate use of the range feature, there is an in-core save area.

Perhaps the most powerful aspect of PEDIT is the ability to define and invoke MACROs. If there is a sequence of key stroke commands that I find myself doing frequently, then the next time I'm ready to use them, I can start by pressing the 'DEF MACRO' key. Then, as I go through the key sequence, it will be saved as a MACRO. PEDIT will ask for a name for the macro, and save it on disk if I choose. If I don't save it on disk, then it will only be available until the end of the current editing session. To use a MACRO subsequently, one merely hits the INV (Invoke MACRO) key followed by the macro's name. One can even invoke a macro from within a macro!

SOME 'WHAT IFS'

— What if the diskette fills up as PEDIT rewrites the buffer to disk when I'm done editing? PEDIT will save the original file with a .BAK extension, and I must either make room for the text on the current diskette, or insert another one before continuing. PEDIT beeps when a diskette is full.

— What if I'm trying to print a defined range on my printer (yes, I can do that too), but I've forgotten to turn on the printer? PEDIT just sits there waiting, without crashing. After I've turned on the printer, I simply execute the print command again (CTRL PRTSC).

— What if I want to execute some DOS system commands while I'm editing? I just press the 'SYSTEM CMNDS' key and can then display a directory, delete files, or change the default drive.

Well, that's a quick look at PEDIT. I find it useful since I do mostly programming on my machine. I have not covered everything in great detail, and there are a few things I did not mention at all. For example, there are other ways to move lines and fields without using the range definition function; there is an alternate way to set tabs; and there is a HELP key when you get stuck.

PEDIT is a great little editor. If you want more information be sure and contact the folks at Satellite Software International. □

vedit

VEDIT

by Frankie Detesto

VEDIT for the IBM PC
price $150
CompuView Products Inc.
1955 Pauline Blvd., Suite 200
Ann Arbor, MI 48103
ph. (313) 996-1299

VEDIT is a very fast screen text editor. It comes in a number of versions for a wide range of computers. If you use several kinds of micros or program both on an IBM PC and a main-frame then you know the difficulty of switching between very different text editors. Perhaps you opted out for a ho-hum editor or word processor that would run on both just so you wouldn't have to learn 2 complete sets of keystrokes. With VEDIT this is a thing of the past. If you have an 8088 8086, Z80, or 8080 you can use the SAME text editor, regardless of whether your machine is memory mapped or hooked up to a CRT. It makes no difference to VEDIT. You can set up the keystroke/command structure any way you like.

The fact that VEDIT covers so many machines is really a sidelight to this review. Most of us only use one computer and it depends on how features are implemented on our own machine that we are interested in. VEDIT handles all the usual text movement needs effortlessly. Moves, deletes, copies to and from save areas (10 of them in memory) are very straight forward. The keystrokes or function keys to accomplish text

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

manipulation can be determined when you customize VEDIT or can be left as they are originally installed.

Paging from one page to the next is FAST. This form of high octane screen flipping makes Wordstar look like Electric Pencil with the power shut-off. Think for a minute how much time you spend bouncing back and forth in a program from one page to another. Usually what you want to look at is several pages apart. Speedy paging is worth its weight in gold and VEDIT is a leading contender in that category. If you press the PageDown key fast enough you can see remnants of previous pages still hanging around. This is a result of good programming. You can easily determine which text editor programmers are speed conscious. The better ones begin rewriting the screen IMMEDIATELY when the Page key is pressed. More archaic editors wait to refresh the complete page before checking to see if you want still another page. The results of this technique are a slight carry over of text while you rapidly (and I mean rapidly) press the paging key, but flipping back and forth through several pages is a dream.

A brief mention of VEDIT's advanced features.

You can customize the keyboard any way you wish except for the return key (which splits the line.) This customizing will allow duplicate commands for the same function. It is possible to setup a function key and a Ctrl-G to both execute the identical command. There are 10 save areas in memory to choose from. You can load any of them from disk. An especially valuable option is the ability to have a VEDIT.INI file that initializes VEDIT when you bring it up. It is just a series of commands (Macro) that determine whatever parameters are likely to change frequently. Meaning you can have one .INI file to set left and right margins for letter writing, another .INI file for your programming diskette to setup no margin and to use a real TAB character instead of spaces for conserving file size. There s an UNDO key that will bring back the last line you deleted if you are still on it. For very large files you have automatic buffering both forward and backward. It is simple to edit a file that is 200K in length, you open the file for input on drive A: and open it for output on drive B:. The usual simple printing is provided, nothing fancy like most text editors.

VEDIT has a Macro language. The command line is on the bottom of the screen. You can set off the ROW, COL, and Buffer information if you wish, this will leave just one line on the bottom of the screen. The ESC key is used to jump in and out of the command mode. The command language is much like many other editors. Single letters are used to denote KILL or MOVE or REPLACE. Here is a sample command that says replace all this's with that's.

#[Sthis$that$]

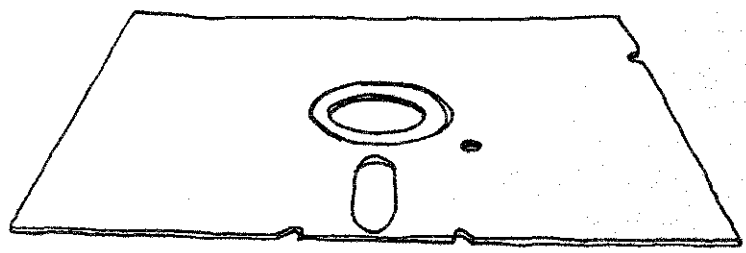
With these sort of macro commands at your disposal you can setup custom searches. With other editors, the search takes you to the FRONT of the word you are trying to find, no options possible. With VEDIT and other sophisticated text editors you can search for an occurrence and then POSITION the cursor where you want it.

The drawback of Macro languages is spending the time to learn them. You only need a few commands for frequently used procedures so this is not really an obstacle. The advantage is that you are limited only by your own creativity. For example, with 15 source files on a diskette you wish to rename a variable to a more appropriate name. However the variable is used extensively in all the files. With a Macro language you can just type in a short set of commands to open, search, replace, and close all 12 files. Go get a cup of coffee while this is going on. If you like freedom and unfettered access then a Macro style text editor is for you.

The documentation is complete. It is a little weak in terms of layout. The information is all there and is very clear, but it could be presented in a more lucid manner. Part of this is a result of serving many varying computers. Every Macro command is shown with examples to illustrate how to use them.

VEDIT is written by a talented programmer who is also a principal of the company. It is reassuring to know that you can found a business on just one excellent piece of software. CompuView has added other products besides VEDIT. They have their own enhanced version of CP/M-86 and a high powered communications program.

VEDIT is a pleasure to use. Other text editors would do well to judge the power of their features in comparison to VEDIT. □



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ASSEMBLER

PROGRAMMING IN ASSEMBLY LANGUAGE

by Art Zachai

The goal of these programs (and ones to follow) are to illustrate and illuminate the information contained in the other ASM manuals. The programs are not guaranteed to do anything other than give you some ideas about how to write your own programs. If you have tried to use the assembler, then you know that using many of the assembler directives is confusing. It is hoped that by showing a few simple routines with a minimal number of directives, you will begin to get a feel for how they are used.

A complete understanding of the architecture of the 8086/88 chip is not essential to programming in assembly language, but the register model shown in the Tech Ref on Pages B-2, -3, and -4 is indispensable. If you intend to do a lot of assembly language programming, the Tech Ref manual is a must.

You will notice that each routine starts with more than a few lines of comments. I have not done this just because I am showing you these routines. I do it in every program I write. I hope I can save you some trouble in the future by suggesting you do this too. It is much easier to have an English description of what the program does and how it does it, than to try and figure it out from reading the code. You will find that programs you have written yourself begin to look like jibberish after only a few months of not seeing them. I have also stressed the use of symbolic constants, and long symbol names. The assembler allows this which makes a program more readable. Good code is code that can be understood by someone other than the author, not code that is tricky and obscure. A good programmer also writes modules that can easily be modified by another good programmer.

Protection of your code or algorithms is easily accomplished by distribution security and copyrights, not by writing obscure code. I have stressed writing short modules. I have not done this just for the sake of these sample programs. I do it all the time. Short routines are easy to read, easy to debug, and can be used in many different programs. This results in an interesting paradox: A complete program which is made up of many small routines is easy to read in the well documented source code, but appears to be a plate of spaghetti if dis-assembled from the run module. I do not feel that one can learn to write good assembly code by dis-assembling BASIC. I do think you can learn from looking at how someone else did it, if they provide the source code, and intended it to be well documented.

The first program shows a skeleton for a DOS program which will do absolutely nothing. But it can be called from DOS, by name, it will load, execute, and return to DOS. You will see the skeleton appear in other programs. When looking at the programs, read the descriptions of the instructions or directives in the assembler manual and try to understand why they are used in the program. Using a cook-book approach (by exactly copying what is done here) will only limit your creativity. □

Note: These listings are part of about 20 programs or routines written on the IBM Personal Computer, using the IBM Macro Assembler. The complete set of ASM tutorial routines is available from Art Zachai for $25. The routines are designed to illustrate the use of the assembler, MS-DOS, and the BIOS ROM. No attempt has been made to use every instruction available in the 8086/88, although some of the more unique instructions and some interesting addressing modes are illustrated. The programs show you how to go about accessing many of the capabilities that you will want to include in your own programs, such as making use of the DOS interrupts, BIOS ROM routines, and internal features of the machine which are transparent to high-level languages.

Three Ways to Learn
Assembly Language Programming
on the IBM-PC
from
ARTHUR ZACHAI

1. If you are proficient in assembly language on another machine, but need to *spin up* on the IBM-PC, attend the one-day **Assembly Language Workshop.** (Eight hours of intensive instruction and hands-on experience. $60.00)

Groups may take advantage of the workshop by providing students (12 maximum), space, and machines (one per 3 or 4 students), and paying a negotiable fee.

2. If you are proficient in a high level language, but new to assembly language, attend the six week **Assembly Language Programming** course. (Six evenings for a total of 18 class hours, with homework. $165.00)

3. If you prefer to learn by yourself, refer to **Twenty Sample Assembly Language Programs.** Useful and instructive routines written for the IBM-PC assembler and DOS, profusely commented and page number references to the IBM manuals. ($25.00)

All three methods will help you to understand:
- Assembler pseudo-op use
- Interrupts and DOS/BIOS structure
- Register use conventions
- Writing "readable" code
- Use of LINK and DEBUG (not in SAMPLE PROGRAM package)

The instructor has over 17 years of professional experience in engineering and computer science, two years with the 8086/88, and 10 years teaching experience at a local university.

Send name, address and phone number for course schedule to:
Arthur Zachai
3817 Veazey Street, N.W.
Washington, D.C. 20016

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
COMMENT*

      Structure for an MS-DOS Compatible Program of the simplest type

              (reserved words are in caps, except in comments)

        Try typing this in, with no YOUR PROGRAM, assemble, link, and
        run it. It will do nothing, but it will assure you that you can
        get to it from DOS and get back to DOS at the end. Try running
        it under DEBUG. Be sure you watch what happens to the stack pointer
        as you single-step through the entire program.

        The word COMMENT treats everything following the next non-blank
        character (*, in this case) as a comment, until the next occurrance
        of that character (which appears at the end of the next line).

-------------------------------------------------------------------------------*

cseg    SEGMENT                 ;NO OPTIONS, THERE IS ONLY THIS MODULE.

        ASSUME CS:cseg,SS:sseg  ;NO LOCAL DATA AREA, NO EXTERNAL ITEMS(DS), AND
                                ; NO STRING OPERATIONS(ES).

prog1   PROC    FAR             ;ASSURES USE OF 4-BYTE RETURN ADDRESS BY THE
                                ; "RET" INSTRUCTION AT THE END OF prog1.

        PUSH DS                 ;ON ENTRY, DS POINTS TO PROGRAM PREFIX CREATED
                                ; BY DOS (SEE DOS MANUAL PG. E-3, 4, & 5.
        SUB AX,AX               ;MAKES AX = 0.
        PUSH AX                 ;THE LAST 4 BYTES ON THE STACK NOW POINT TO THE
                                ; BEGINNING OF THE PROGRAM PREFIX (DS:0).
;  ____________________________________________
; |                                            |
; |          PUT YOUR PROGRAM HERE             |
; |____________________________________________|

                ;NOTE THAT A BLANK LINE IS TREATED AS A COMMENT, NO ";" NEEDED.

                ;BE SURE STACK IS BACK TO WHERE IT WAS AFTER THE PUSH AX ABOVE,
                ; AND ALL SEGMENT REGISTERS ARE AS THEY WERE WHEN PROG1
                ; WAS ENTERED.

        RET     ;CAUSES A JUMP TO START OF PROGRAM PREFIX, WHERE DOS
                ; LEFT AN INT 21, WHICH RETURNS YOU TO DOS (SEE DOS APPENDIX E).
        ENDP    ;END OF PROCEDURE

cseg    ENDS    ;END OF SEGMENT

;.....................................................................

sseg    SEGMENT STACK           ;SEGMENT WITH COMBINE TYPE OF STACK, SEE
                                ; "LINK" IN DOS MANUAL.
        DW  2  DUP(?)           ;UNINITIALIZED STACK AREA. ALLOW ONE WORD FOR
                                ; EACH "PUSH" IN A STRING OF PUSHES WITHOUT POPS,
                                ; PLUS 80H FOR ANY DOS OR BIOS CALLS.
sseg    ENDS                    ;THE LINKER/LOADER WILL INITIALIZE THE STACK
                                ; SEGMENT AND POINTERS TO THE HIGHEST ADDRESS
                                ; IN sseg SEGMENT.
;.....................................................................

        END  prog1              ;ALWAYS NEED AN END, AND CAN BE USED TO
                                ; SPECIFY THE START ADDRESS OF THE MODULE.
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
The IBM Personal Computer MACRO Assembler 01-23-83                PAGE    , ,
DUMB TERMINAL


                NAME DUMB_TERM_1
                TITLE DUMB TERMINAL
                PAGE 64,132
        COMMENT*
                THIS PROGRAM MAKES THE PC ACT LIKE A DUMB TERMINAL

                PROGRAMMER:      ZACHAI
                DATE WRITTEN:    3-7-82
                CHANGE HISTORY:

         DESCRIPTION:

                A DUMB TERMINAL IS IMPLEMENTED USING THE ASYNC-COMMUNICATIONS BOARD.
                THE UART MUST HAVE BEEN SET UP FOR THE PROPER BAUD RATE, PARITY,
                BITS/CHARACTER AND STOP BITS, PRIOR TO ENTERING THIS PROGRAM.

                THE TERMINAL IS IMPLEMENTED USING A "POLLED" TECHNIQUE. THE SERIAL
                PORT IS CHECKED TO SEE IF A CHARACTER IS READY, AND IF THERE IS,
                IT IS READ AND DISPLAYED ON THE SCREEN. IF NO CHARACTER IS READY
                AT THE UART, THE KEYBOARD IS CHECKED FOR CHARACTER READY, AND IF
                THERE IS, IT IS READ AND OUTPUT TO THE SERIAL PORT.

                THE ONLY CHARACTERS GOING TO THE SCREEN CAME FROM THE SERIAL PORT,
                HENCE THIS IS A FULL DUPLEX TERMINAL.

                THE PROGRAM USES DIRECT I/O TO THE UART AND BIOS CALLS TO ACCESS
                THE KEYBOARD AND SCREEN.

                THERE IS NO MODEM CONTROL IMPLEMENTED. THE ONLY SIGNALS NECESSARY
                FOR DATA COMMUNICATIONS ARE TX-DATA, RX-DATA, AND SIGNAL GROUND.

         ENTRY/EXIT PARAMETERS:

                ENTRY IS FROM DOS BY TYPING:     A)TERM1

                EXIT IS BACK TO DOS BY TYPING  CTRL-F1.

         SUBROUTINES CALLED:

                NONE.
        ..................... STACK AREA ..........................................*

        SSEG    SEGMENT STACK   'STACK'
                DW 12 DUP(?)    ;LOOK THROUGH THE BIOS ROUTINES USED TO FIND STACK


                                ;SIZE NEEDED FOR ROUTINES USED, ADD TWO FOR THIS PROG.
        SSEG    ENDS
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
PAGE
;..................... EQUATES ..............................................*

VIDEO_IO_INT     EQU      10H      ;BIOS INTERRUPT -SEE TECH.REF. PG.A-43
TTY_OUT_FCTN     EQU      14       ;BIOS CALL PARAMETER FOR INTERRUPT 10
KBD_IO_INT       EQU      16H      ;BIOS INTERRUPT -SEE TECH.REF. PG.A-23
KBD_STATUS_FCTN  EQU      1
KBD_READ_FCTN    EQU      0
STATUS           EQU      3FDH     ;UART STATUS PORT (SEE TECH.REF. PG.2-125)
DATA             EQU      3F8H     ;UART DATA PORT
RX_RDY           EQU    00000001B  ;STATUS BIT = DATA READY (TECH.REF. PG.2-137)
TX_FULL          EQU    00100000B  ;STATUS BIT = TX BUFFER EMPTY

;..................... THE ROUTINE ..........................................
CSEG     SEGMENT
         ASSUME CS:CSEG,SS:SSEG,DS:CSEG
TERM     PROC FAR
         PUSH DS             ;SET UP FOR RETURN VIA DS:0.
         SUB AX,AX
         PUSH AX

         SUB BX,BX           ;LEAVE A ZERO IN BX TO MAKE THE DISPLAY ALPHA MODE
                             ; SEE TECH REF. PG.A-44
CHECK_UART:
         MOV DX,STATUS       ;UART STATUS PORT ADDRESS TO DX.
         IN AL,DX            ;GET UART STATUS.
         TEST AL,RX_RDY      ;CHECK FOR DATA READY (ERRORS NOT CHECKED).
         JNZ READ_UART       ;IF READY, GO GET THE DATA AND DISPLAY IT
         AND AL,TX_FULL      ;ELSE CHECK FOR TX BUFF RDY
         JZ CHECK_UART       ;GO GET STATUS IF NOT RDY,LAST CHARACTER NOT OUT YET.

         MOV AH,KBD_STATUS_FCTN  ;KEYBOARD CHECK
         INT KBD_IO_INT
         JZ CHECK_UART       ;IF ZF=1 NO KEY READY, SEE TECH.REF. PG.A-23,LINE 1648.

         MOV AH,KBD_READ_FCTN
         INT KBD_IO_INT      ;GET THE KEY CODE
         AND AL,7FH          ;MAKE 7 BIT ASCII(ASSUMIG ITS AN OK KEY)
         JNZ SEND_DATA       ;CALL IT A VALID CHARACTER IF NOT ZERO, THE "AND AL"
                             ; PRIOR TO THIS JUMP SET THE CONDITION CODE IN FLAG REG.
         CMP AH,5EH          ;IF AL=0 CHECK FOR CTRL-F1, TO EXIT (AH=SCAN CODE)
         JNZ CHECK_UART      ;OR SOME OTHER BAD KEY, DON'T SEND IT
         RET                 ;ELSE EXIT

SEND_DATA:
         MOV DX,DATA         ;UART DATA PORT ADDRESS TO DX.
         OUT DX,AL           ;SEND CHARACTER
         JMP CHECK_UART      ;THEN CHECK FOR INPUT CHARACTER WAITING.

READ_UART:
         MOV DX,DATA                 ;UART DATA PORT ADDRESS TO DX.
         IN AL,DX                    ;GET CHAR FROM UART
         MOV AH,TTY_OUT_FCTN         ;TTY TYPE OUTPUT
         INT VIDEO_IO_INT
         JMP CHECK_UART

TERM     ENDP
CSEG     ENDS
         END  TERM           ;END OF MODULE, START ADDRESS IS "TERM"
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

pj interview

PJ PROFILE

An interview with Ken Smolek

PJ: When did you start working with computers?

KEN: 20 years ago this January, that was in 1963. I was in high school, it was a National Science Foundation Math camp at Florida State when I was 15. We played on an old IBM 709. An unbelievable 2 acres of computer which ran Fortran 2. Then I did a few things on computers in college but was not too serious about it. I did get a degree in Physics. Computer Science was still very much a part of the electrical department. My first real experience was between my junior and senior year when I got a job with a company that was doing process control work on minis. We had to write our programs in PDP-8 assembly language. There were few machines dumber than that one. It had a 12 bit word 4K was all it could address. (Total of 16K memory.)

PJ: Only being able to address 4K seems almost ridiculous any more.

KEN: I guess, but it worked well for what we needed it to do. We made thread rolling dyes for making screws. You take 50 planar and circular surfaces and do the calculations for the dye. Most of my past computer projects and even some of my recent work has been in process control with minis. I didn't do any main-frame work until 1977 when I started working for the local county data processing plant department.

PJ: How long have you been consulting?

KEN: Off and on since I left college. After the PDP-8 work there were always jobs floating around or people calling in seeing if I was interested in a particular project.

PJ: When did you get your first personal computer?

KEN: I didn't really have any interest in 8 bit micros for a long time. The instruction set was just a little too ugly for me to invest much long term effort in them. I really wanted something that was fun to program. Certainly every chip has its quirks, but you need to be able to add registers to memory and add memory to registers so that you enjoy the process. On the 8 bit chips you always have to worry about moving values into a register before you can do anything with them.

PJ: Which micro did you finally pick?

KEN: Seattle Computer finally came out with a 16 bit system. Their 8086 8 megahertz computer. They were the only people selling 16 bit wide memory so I bought one with 64K and two 8 inch disk drives. This was in December of 1979.

PJ: What language did you use on it?

KEN: They did not have even a BASIC interpreter at that point so all programming was done in assembly. BASIC is not a favorite of mine so that was not a drawback. I did not have a complete operating system either, I was using an early version of MS-DOS, it was really Seattle Computer Products 8086 DOS. I also did not buy the disk controller they were selling so I had to write my own, or really modified their version of one. It involved a lot of hex coding and hand assembly. My first version of the BIOS was done in the same way.

PJ: Did they have any utilities for it yet?

KEN: They had a symbolic debugger which I still find very useful. When the system comes up it is in a monitor which does not do disk reads and writes but it does do boots. This allows you to read and write directly into

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
/*      C  Language Version (Computer Innovations CI-86)          */
/*      Program to initialize Microline 82A printer operating     */
/*      modes : CPI, LPI, and Line Length                         */
/*      by Ken Smolek                                             */
/*      created          5/24/82, 2115 hrs                        */
/*      last modified    2/20/83, 0945 hrs                        */

#define ESC     27
#define GS      29
#define RS      30
#define US      31

main()
        {char   c;
        bdos(5,'\r');                   /* issue a CR/LF */
        bdos(5,'\n');
        printf("\nInitialization for Microline 82A Printer\n");
        printf("Characters/Inch (1=5, 2=8.3, 3=10, 4=16.5)? ");
        c=getcrt(c);
        putcrt(c);
        switch (c)
            {case '1':                  /* 5 CPI */
                bdos(5,RS);
                bdos(5,US);
                break;
            case '2':                   /* 8.3 CPI */
                bdos(5,GS);
                bdos(5,US);
                break;
            case '4':                   /* 16.5 CPI */
                bdos(5,RS);
                bdos(5,GS);
                break;
            default:                    /* 10 CPI */
                bdos(5,GS);
                bdos(5,RS);
            }
        printf("\nLines/Inch (1=6 lpi, 2=8 lpi)? ");
        c=getcrt();
        putcrt(c);
        if (c=='2')                     /* test for 8 lpi */
            {bdos(5,ESC);
            bdos(5,'8');
            }
        else
            {bdos(5,ESC);               /* default is 6 lpi */
            bdos(5,'6');
            }
        printf("\nLine length (1=long line, 2=short line)? ");
        c=getcrt();
        putcrt(c);
        if (c=='2')                     /* test for short line */
            {bdos(5,ESC);
            bdos(5,'B');
            }
        else
            {bdos(5,ESC);               /* default is long line */
            bdos(5,'A');
            }
        bdos(5,'\r');                   /* issue a CR/LF */
        bdos(5,'\n');
        exit(0);
        }


/* get and put to crt */

getcrt()
        {return (bdos(7));
        }

putcrt(c)
        char    c;
        {bdos(6,c);
        }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOOLS

by Greg Estes

Two topics will be covered this time, Self-Documenting programs and an alternative to the "3-Deep" method of wading through a system.

SELF-DOCUMENTING PROGRAMS:

The concept of self-documenting programs is a good one, but is it really possible? We all know the feeling of trying to write up the documentation for a system once we have completed the programming. Established system designers tell us we are supposed to do it first, but that rarely happens, especially on smaller applications. As programmers we often end up trying to document how the system works for someone else who will create the final manual for users. The difficulty lies in the fact that the system is all OLD HAT to us and we make large jumps in logic when we try and recreate the step by step guide of what is going on. It is very boring to us so we rush through it. The documentation we need the most is the program flow and file usage charts so we can go back and maintain the thing. Here is a sure fire way to accomplish this system documentation with the least amount of effort (only explain things once to yourself.) We will describe the method briefly because it is quite simple.

The first time you enter your text editor to start writing a program you type in the appropriate headings. (The lines which follow are all COMMENT lines that are REMed out.)

```
rem PROGRAM NAME : Master file maintenance
rem FUNCTION     : To compress unused records
rem AUTHOR       : Bobby Speedcoder
rem CREATED      : 4/4/83
```

. . .etc

You know, the usual stuff. But before you head off into initialization code put in just a few more lines.

```
rem FILES USED:   CUSTMAST.DAT - Updated
rem               CUSTINDX.DAT - Updated
rem               DELPRINT.PRT - Printout
rem               COMMON.DAT - Passed Parameters
```

. . .etc

You should expand these comments as much as you consider necessary to cover the important files, #include files, and so on. If there are several programs in your application this would be a good time to flesh out your logic flow by doing some pseudo-code lines.

```
rem initialize variables
rem open files
rem mark deleted records
rem write deleted report
rem compress out deleted records
rem rebuild index file
rem close files
rem spool report
rem exit program
```

This outline should be done for each program you plan on writing. Then when it comes time to write the actual lines of code you just "code between the lines." The actual GOSUB labels immediately follow the comment lines.

Most of us know we are supposed to write this sort of outline first but often just skip it. It is just as easy to write a code module and THEN write the REM line to indicate what it does. Most us put the usual header stuff in our programs but leave off the file name list. If we can force ourselves to just put one comment line proceeding each subroutine LABEL we are home free as far as the Self-Documenting process is concerned.

Now we have 10 or 20 programs that are appropriately commented, we are ready to move on to another project, our fellow workers need the documentation (and we will need it 6 months down the line.) What to do? Just write a quick program to take each source file one at a time and PRINT OUT all of the comment lines. That is about as close as we will ever get to self-documenting programs. The result is a system manual that in detail describes every program with the file usage and logic flow explained. When it comes time to do any maintenance (by you or someone else) there is an excellent chance that the problem can be pin-pointed to the exact subroutine before you ever look at a source listing. Besides providing a complete system notebook it serves as a guideline for the writer of the user manual.

Enough said, I think you see the possibilities.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3 DEEP:

Many systems we use follow the "3 deep" method of hooking programs together. It is simple and as long as you don't go more than "3 deep" you do not get lost. There are other ways to link one program to another. After all, what are these menus for? A user wants to get from one WORKING screen to another.

Here is your basic MASTER MENU:

1. CUSTOMER MAINTENANCE
2. VENDOR MAINTENANCE
3. INVENTORY MAINTENANCE
4. SYSTEM UTILITIES

If you chose option 1 then you arrive at another menu:

CUSTOMER MAINTENANCE

1. ADD CUSTOMER
2. DELETE CUSTOMER
3. CHANGE CUSTOMER
4. PRINT/DISPLAY

Only after picking one of these options do we arrive at our WORKING screen to accomplish the work we started out to do. But probably you were at some other working screen before you decided to go into customer maintenance. This means you had to back out of 2 menus to get to the main menu and then made 2 more decisions to arrive at your destination. This is not much fun for an experienced user, wading around multiple menus, and if you are a programmer that is TESTING between several menu options it is down right excruciating.

What would we really like to do? How about go from one working screen DIRECTLY to another working screen. Do not pass go and do not collect 200 menu options.

First design a MASTER menu with everything on it. All options except the working screens themselves. Group the options by categories as you would in a normal menu approach. Number them, you will note, in blocks incremented by 10.

MASTER MENU

CUSTOMER MAINTENANCE	INVENTORY MAINTENANCE
1 ADD CUSTOMER	21 ADD ITEM
2 DELETE CUSTOMER	22 DELETE ITEM
3 CHANGE CUSTOMER	23 CHANGE ITEM
4 PRINT/DISPLAY	24 PRINT/DISPLAY
VENDOR MAINTENANCE	**SYSTEM UTILITIES**
11 ADD VENDOR	31 SYSTEM OPTIONS
12 DELETE VENDOR	32 BACKUP FUNCTIONS
13 CHANGE VENDOR	33 PASSWORD MAINT
14 PRINT/DISPLAY	34 EXIT TO DOS

The Master Menu, then, serves as a complete naming reference for every working screen you have with a distinct number to access it.

With every working screen having a unique number, we can write a VERY SMALL program that loads quickly and merely chains to the working screen we want. We could call it our Grand Junction program. It will look something like this.

```
.
.
.
100 REM RECEIVE THE COMMON VARIABLE AND THEN
110 IF OPTION>0 AND OPTION<5 THEN CHAIN "CUSTOMER"
120 IF OPTION>9 AND OPTION<20 THEN CHAIN "VENDOR"
130 IF OPTION>19 AND OPTION<30 THEN CHAIN "INVENT"
140 IF OPTION>29 AND OPTION<30 THEN CHAIN "SYSTEM"
150 END
```

The Master Menu just chains to this program too. Every other program uses it also. Being small there is little overhead.

One other thing is needed. Once you arrive at a working input screen, and are ready to leave for another one then you ask the user a question. Let us assume that hitting the ESC key at the first field in a screen signals "time to exit to somewhere else." When the user presses the ESC key at the top of the screen you need only prompt them with the something along the lines of:

```
        ADD CUSTOMER

NAME  :

ADDR  :

CITY  :

STATE :

ZIP   :




Enter Option number, or ESC to Main Menu ←
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

They have 3 choices, entering the Master Menu number they want which will then call the Junction program which chains to the appropriate program, they can hit the ESC key again to take them to the Master Menu if they have forgotten the option number, or they are allowed to press the RETURN key to remain in their current screen because they didn't really want to go.

The advantages of this system are as follows:

1. Novice users always have the menu to go back to.
2. Experienced users don't EVER have to bother with the Main Menu again.
3. Testing time is greatly reduced for programmers just as it is for experienced users. You move from one screen to the next with not intermediate steps.
4. Having everything on one Main Menu is a tremendous selling point. This allows you to create a fancy bells and whistles Menu that takes a while to load but looks sharp, and you need not worry about slowing down experienced users. Imagine a buyer looking at your system with 50 options staring them right in the face, and then he goes to another booth and looks at 4 choices for a master menu. Which one would you choose?
5. Security is easily maintained because all users must pass through one junction program.

If you have any doubts about this approach, and do not consider it a marked improvement I would suggest trying it briefly. It is like going from WordStar to the Personal Editor.

Programming is a perpetually developing art form. We do not view this menu layout method as an end in itself. Let us know of any techniques you might be using in your own systems so that we can share them with other programmers.

MISC:

A few gems from the PC Monitor Newsletter. This one is a single line UPPERCASE function as well as single line MAX/MIN function. Written by Harry Foxwell.

```
DEF FNUCS$(A$) = CHR$(ASC(A$) + 32*(A$>="a"
    AND A$<="z"))
```

In other words, if A$ is larger or equal to "a" AND A$ is less than or equal to "z" then we know it is lower case, which means the expression evaluates to a TRUE which is of course a negative one. Thus 32 times -1 equals a minus 32 which is the number needed to make an lowercase letter a uppercase letter. If the expression is false then it is zero and we leave A$ untouched.

```
DEF FNMAX(X,Y) = X*ABS(X>=Y) + Y*ABS(X<Y)

DEF FNMIN(X,Y) = X*ABS(X<=Y) + Y*ABS(X>Y)
```

Both of these functions illustrate a creative use of expression evaluations. If you use logical expressions with AND and OR inside of brackets then the result will be either zero or a negative one. Used in this manner you can design many creative expressions. Let's say you had to determine whether a person was male or female and in your application every calculation would be changed accordingly.

```
10 TRUE = -1 : FALSE = 0
20 ... find out if they are female
30 IF ANSWER = "y" OR ANSWER$ = "Y" THEN
   FEMALE = TRUE ELSE FEMALE = FALSE
40 ... so FEMALE is either 0 or -1
```

Old method of determining values:

```
100 IF FEMALE THEN WEIGHT = 100 ELSE
    WEIGHT = 170
```

New way:

```
100 WEIGHT = 170 + (FEMALE*70)
```

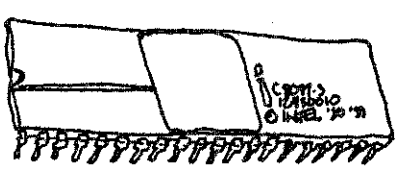
Old Way:

```
200 IF FEMALE THEN COUNT = 20 ELSE COUNT = 10
210 FOR I = COUNT TO COUNT + 5
```

New way:

```
210 FOR I = 20 + (FEMALE*10) TO 25 + (FEMALE*10)
```

□



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

I'LL BE C'ING YOU

by Dr. Neil Bennett

The way to get the absolute best performance out of your micro is to write programs in assembler. Sometimes the only way to make peripherals sing and dance in time to the music is to drive them from an assembler routine. However, if you are writing a large program which is going to be around for a few years, assembler may not be the best possible choice. The micro market is changing so rapidly that a few years is an eternity. (Look back a few years; the changes in the future will be even bigger). There will still be machines that can run your assembler code, but there will be other machines that you would like to have your program running on as well. You could rewrite your code in assembler for the new machines, but that gets old quickly.

At this point I feel I should confess my bias. I am writing software for the market place and my income depends on appeal of the program and the size of the potential audience. As a counter-example, I would not be tempted to write a program to calculate the eclipses of the sun for the next 500 years. If the program was for a PC and needed 512KB of memory and the 8087 chip, then I would grow thin off the royalties.

I want to be able to write a system for the present breed of microcomputer and then, when bigger better faster machines come along, be able to crank the handle and put the system on a new machine. I don't want to spend 4 months building the system today and then another 4 months rebuilding it for a new machine.

I can't prove it, but I think the C language is about as perfect for my needs as I am going to find. There is just one reference book on C, and that's by Kernighan and Ritchie (known as K&R). The language went through a long incubation period within Bell Labs before surfacing in the outside world. Approximately 94% of the UNIX operating system is written in C with the rest in assembly to talk to peripherals. About a year ago I noticed the first C compilers appearing on the micro market for 16 bit machines. I purchased the Telecon C compiler last August and settled down to write my programs in C. The idea was to write my program once and then it could go onto all sorts of machines. The K&R list of machines having C systems: PDP-11 (the original implementation), Honeywell 6000, IBM 370 and Interdata 8/32 (and that book was written in 1978!.) There are C compilers for lots of other machines such as the 8080, Z80 and of course the 8086 (which we all know and love in it's 8-bit form as the 8088).

I write code in a simple style. Even so, I found I was constrained by the very simple code structures in C UNTIL I tried doing things their way. I now find I can put a C program together easily, and it is far more readable than my assembler code ever was. The very simple structures force you to write simple functions. There was an unexpected bonus with C. When I rewrote a large program (HHB) that I had written in 6502 assembler for the APPLE, I found there was all sorts of near-duplicated code. Each time I had written a similar function in assembler, I had done it in a slightly different way. Because the assembler source was so big I was not able to see the near-duplications. In C, you have to work at it to write code to do similar tasks in more than one way and because the overall system was less lines of source, it was easy to spot the duplications and build one function to replace several.

C allows you to concentrate on what the program will do rather than the mechanics of how it is to do it. Now, I don't know which registers are being used and, what's more, I don't care. But being an old assembler buff I could not resist the temptation to look at the assembler code generated by the C compiler. I wish I hadn't. I estimated that the code generated was about 3 times bigger than I could do by hand. The only explanation possible was that the C compiler treated the 8086 as just a 16-bit wide Z80; that this compiler was originally written for an 8-bit machine and upgraded to exploit the IBM PC market. However, the programs worked and I pushed on. The original HHB program ran on a 48KB APPLE and I started getting worried when my IBM PC C-based system passed 64KB with more to go. I have 128KB so I can keep on building; the reality of the market place is that there is a bigger market for a program that will run in a 64KB machine rather than a program which needs 128KB. At this point I started thinking there must be better C compilers out there. I also started thinking I could build my own C compiler more easily than translate the C-based HHB program into assembler by hand. (Particularly, if I wanted to make a few changes to HHB in the tuning stages.) So, I held my nose and pressed on to finish the C-based HHB and not look at how big the code space was getting.

As I write this (Jan 13th,1983), my own C compiler for the PC is about 70% finished. The design is about 95% complete and there is lots of tedious clerical work left to generate assembler code for each operation dealing with about 5 flavours of operands. My compiler generates one instruction to increment a counter. The Telecon system, at worst, generates 5 instructions. I also plan to backend the system with a peep-hole optimizer. This is an exotic name for a simple concept. The assembler code will be

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

buffered to about 30 lines before going out to disk. Simple heuristics can be used to find curable inefficiencies in this "peep-hole" . All that looks like a mouthful. Here's an example. If the instruction about to go out to disk is a jump, and its target is in the peep-hole, then the jump can be made into a jump short (because the target is within 127 bytes). Another example; to cope with all situations the compiler has to generate code for an "if" like this:- je cc1; jmp cc2; cc1: do some code; cc2: common paths again. If the optimizer finds the cc2 label in the peep-hole, then the code can be simplified to :- jne cc2; cc1: do some code; cc2: etc.

I have used the same register conventions as the Telecon system. This allows me to build a mixed program with some functions compiled by the Telecon compiler and other functions compiled by my own C compiler. Such an approach has other advantages in proving out functions one by one.

The C Compiler itself is written in C. Can you imagine a BASIC system written in BASIC? It's easy to build a C system in C and there are some definite advantages to doing it that way.

Here's the first. I am building my C compiler using the Telecon system. When I am all through I can use my own C compiler to compile itself. That will result in my C compiler being much smaller, but still generating the same optimized code. Because it generates fewer instructions, it will also run faster.

Here's the second. You can "migrate" the compiler onto another machine. We will both get confused if I don't use a concrete example. The obvious machine that I will want to move to is the APPLE 68000-based LISA. Most C compilers have a distinct code generation stage. This extra step is part of their inefficiency - the assembler code is generated at the very last. To migrate to the 68000, I would change the C system on the 8086 to generate 68000 code. I would then have a 8086 based system I could use to compile C programs to run on the 68000, i.e. a cross compiler. I could then compile the new C compiler on the 8086 cross compiler. This would give me a C compiler which would run on the 68000 and it would generate 68000 code. Sounds simple? This is the migration path that was probably used to move from the Z80 to the 8086. It works, but if the original C compiler was written especially for the Z80, the migrated 8086 system might not be very good. The best way to get a well matched C compiler for the 8086 is to start from scratch with that machine in mind.

Those are a few of the arguments for the assembler buffs. If you write in BASIC why should you consider C? In a recent Creative Computing, Will Fastie compares the execution times of BASIC and C (and others). The infamous Sieve benchmark takes 2020 seconds using the BASIC interpreter and the Lattice C takes 11. That's a ratio of about 184:1!. Obviously that's not comparing like with like (an interpreter vs. a compile and assemble); I suggest you read the article and also the original Sieve benchmark article and it's followup.

What happened to the option of seeing what other C compilers had to offer? This turns out to be rather expensive. The Lattice C compiler (or others) may be what I want. To sample it would cost me $500 and if it is just what I need everything would be dandy. But if not, then it would be a rather expensive experiment.

EPILOGUE:

This is a look back on that "C" project (it is March 31st, 1983 now.) For my purposes it is complete.

The overall concept was to build a compiler which would generate compact code. I achieved a 3 to 1 assembler instruction saving (vs. Telecon), but after assembling my programs the ratio of the code-space came out at 55%. The memory reference instructions have to be there and they take up 3 or 4 bytes per instruction.

The optimizer proved to be much more efficient as a separate system when the entire target program could be held in memory.

Compiling my HHB program, I found that it was still not small enough to fit in the 52KB left by the operating system in a 64KB machine. It was however, small enough to fit into a 64KB machine without the operating system. So, I went through the exercise of changing the program so that it communicated directly to the BIOS in ROM, and built a convoluted loading system to overlay the operating system. It works but it's messy.

Even though I had found a way to shoehorn the program into a 64KB machine, I was not happy with the result. I would much prefer to assume a 128KB machine as standard, go back to the standard calls through the operating system and include more features. A couple of weeks ago, IBM introduced DOS 2.0. The resident part of the operating system has gone up from 12KB to 24KB. I asked my friendly computer salesman what this would mean to people with programs in BASIC which used up all the memory in a 64KB machine and his answer was "They would need to get more memory" (IBM stands for Install Bigger Memory). So, it seems the trend towards 128KB is happening faster than I thought. [See also an article "Hard Choices for Software Houses" BYTE April, 1983 p. 242].

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

So, writing my own "C" compiler did not solve my problem; it just went away! However I find myself in a stronger position in 3 ways. Firstly, I know a lot more about "C" than I did before. Secondly, I have a C compiler which generates very compact code. From a software author's standpoint this is fairly important. The result being that a given amount of source code takes up less room and also with a given amount of memory more source code fits in.

My next programming project will be written in "C" and the code will probably take up more than 64KB. With my own compiler I can make modifications to the source and generate code which makes use of long calls, rather than being restricted to a 64KB limit by the short calls.

REFERENCES

The C programming Language, by Brian W. Kernighan and Dennis M. Ritchie. Published by Prentice-Hall.

Telecon Systems, 1155 Meridian Avenue, Suite 218, San Jose, CA 95125. See their ad. in January, 83 BYTE page 218.

Creative Computing, February 1983, ibm images by Will Fastie, page 278.

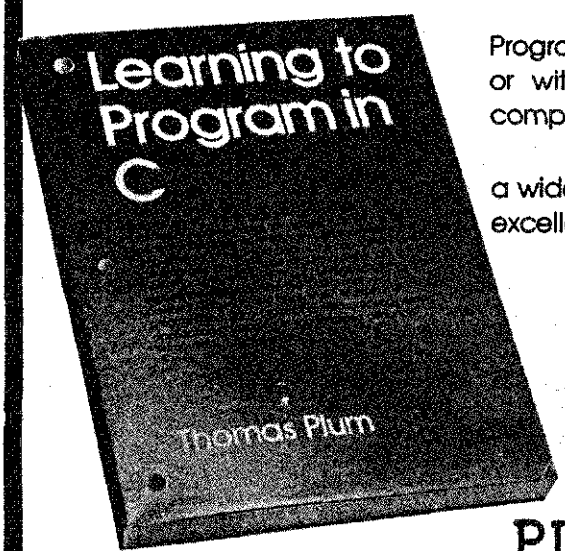
A High-Level Language Benchmark, by Jim Gilbreath, BYTE September 1981, page 180.

Eratosthenes Revisited, by Jim Gilbreath & Gary Gilbreath, BYTE January 1983, page 283.

Lattice C. Lifeboat Associates, 1651 Third Avenue, New York, N.Y. 10028. Also see PC Magazine, September 1982 (The PC Product Guide issues, page 214. I counted 7 C compilers in this section.)

□

From Plum Hall, an introductory Book on C.



It has been several years in the making and now it is here. Learning to Program in C, by Thomas Plum, teaches C language from the ground up. With or without previous programming experience, anyone acquainted with computers will find a clear description of how C works.

You will find guidelines for writing portable programs that will run on a wide variety of modern computers — micro, mini, and mainframe, with excellent efficiency in all these environments.

Topic areas include:

- **Environmental details - starting C**
- **Data and variables - using the memory**
- **Operators and expressions - intuitive reasons for C precedence.**
- **Control structure - readability rules**
- **Functions - print and scan made easy**
- **Case study - full Blackjack source, from design to documentation**
- **Pointer, struct clarified**

PLUM HALL 1 Spruce Ave, Cardiff, NJ 08232
Phone orders: 609-927-3770

- **explains C step-by-step**
- **practical "how to" approach**
- **describes what happens inside the computer**

372 pp, 7½"x10", Price $25.

☐ Send information on Plum Hall Seminars on C and UNIX™

☐ Check
☐ Mastercard ☐ Visa
☐ American Express

Exp. Date _____

Please send me _____ copies of "Learning to Program in C" at $25. each. (N.J. residents add 6%) enclosed find $ _____

NAME _____
COMPANY _____
ADDRESS _____
CITY _____ STATE _____ ZIP _____
Card No. _____
Signature _____

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ASM LINE DRAWING PROGRAM

Dear PJ,

I am quite impressed with the editorial content of the the first 2 issues of Programmer's Journal. It looks to be a good forum for the exchange of technical information.

I am enclosing two listings of a line-drawing program written in 8088 code. The first is the line drawing subroutine itself and the other is a simple "driver" for testing it. The comments should be enough to explain how to use it as a general-purpose line drawing subroutine. The logic is complex and is best explained by the references listed in the comments.

Besides being very fast (no multiplication is involved), the program offers a couple of options that come in real handy. In addition to the color parameter and the start and end points of the "physical line" , the program expects values to indicate the start and end points of the "logical line" .

This means that the routine will plot the points between (X1,Y1) and (X2,Y2) , but has the capability of stopping before it gets to the (X2,Y2) or can continue PAST (X2,Y2). Thus the routine will plot a subset or superset of the collection of points that define a line.

By repeatedly drawing the "same line" but with an increasing SKIP value and LENgth, while keeping the difference constant, a "laser bolt" can be drawn.

Also, lines offset by various angles can be drawn without knowing where they must end — just give endpoints that are only a few points apart then draw a line that extends past the end point. For example, when X2 = X1 + 1 and Y2 = Y1-1, the line drawn will be 45 degrees from horizontal. When X2 = X1 + 1 and Y2 = Y1-2, then the line will be at an angle 60 degrees from horizontal.

Although I haven't included it here, I have written a dot-plotting subroutine that puts that of the ROM BIOS to shame. The overhead involved in the video interrupt is ghastly! If a dot-plot routine "knows" it will write to the medium resolution graphics screen and uses register shifts instead of MUL to calculate an address, then it will run scores of times faster than the BIOS equivalent.

I hope that you can use these listings.
Thanks for your time.

Sincerely,
Dan Rollins

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        page ,132
        public drawline

;        Dan Rollins (213) 246-5021
;
;8088 self-modifying program implements fast-vector algorithm
; described by Michalsky, Doctor Dobb's Journal #74, 12/82
; see also: FAST-LINE DRAWING TECHNIQUE, BYTE, Aug 81
;
;routine expects to be called with DS:SI pointing
; to a list of 2-byte arguments:
;
;       si+0    = X1    starting clm (0-319)
;       si+2    = Y1    starting row (0-159)
;       si+4    = X2    ending clm
;       si+6    = Y2    ending row
;       si+8    = color  (0,1,2,3)
;       si+10   = length
;                 0     = draw entire line
;                 else = draw sub- or super-set of this vector
;       si+12   = skip length
;                 number of pels to go before starting to draw
;                 0     = draw entire line

;destroys all registers except si and segment regs

code    group   cseg
cseg    segment public 'code'
        assume  CS:cseg, DS:nothing, ES:nothing

; make it easier to access variables and arguments

X1      equ     word ptr [si]
Y1      equ     word ptr [si+2]
X2      equ     word ptr [si+4]
Y2      equ     word ptr [si+6]
COLOR   equ     byte ptr [si+8]
LEN     equ     word ptr [si+10]
SKIP    equ     word ptr [si+12]

; these are values that will be overlayed in the code

INC_X   equ     41H
DEC_X   equ     49H
INC_Y   equ     42H
DEC_Y   equ     4AH

; these are the addresses where new code is overlayed

ADJ_LONG_AXIS   equ   byte ptr cs:[di]
ADJ_MASTER      equ   word ptr cs:[di+3]
TEST_MASTER     equ   word ptr cs:[di+7]
ALT_ADJ_MASTER  equ   word ptr cs:[di+13]
ADJ_SHRT_AXIS   equ   byte ptr cs:[di+15]

drawline proc   far

        mov     bl,INC_X  ;assume Xstep = +1
        mov     ax,X2
        sub     ax,X1
        jge     dl1       ;if X1 (= X2 then no change

        mov     bl,DEC_X  ;Xstep = -1
        neg     ax        ;Xdist = abs(Xdist)
dl1:
        mov     cx,ax     ;save Xdist

        mov     bh,INC_Y  ;assume Ystep = +1
        mov     ax,Y2
        sub     ax,Y1
        jge     dl2       ;if Y1 (= Y2 then no change

        mov     bh,DEC_Y  ;Ystep = -1
        neg     ax        ;Ydist = abs(Ydist)
dl2:
        mov     dx,ax                       ;save Ydist
        mov     di,offset cs:modify_base ;point to the code
                                            ; to modify
        cmp     dx,cx                       ;determine longest axis
        jge     dl3                         ;Y is longer, so skip

        xchg    cx,dx             ;swap Xdist, Ydist
        xchg    bl,bh             ;swap INC/DEC X/Y values
dl3:                              ;modify:
        mov     ADJ_LONG_AXIS,bh  ; the 1st INC/DEC code
        mov     ADJ_MASTER,cx     ; main duty master adjustment
        shr     cx,1              ;set up cycle tester
        mov     TEST_MASTER,cx    ; test for cycling
        mov     ALT_ADJ_MASTER,dx ; alternate adjustment
        mov     ADJ_SHRT_AXIS,bl  ; alternate INC/DEC code

        mov     di,dx           ;DI is counter: long axis length
        cmp     LEN,0           ;if length arg ) 0
        je      dl4
        mov     di,LEN          ; then use it as counter
dl4:
        mov     bp,SKIP         ;get skip count
        mov     cx,X1           ;  and other args
        mov     dx,Y1
        mov     al,COLOR
        xor     bx,bx           ;duty master starts = 0
```

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;---------- top of vector plotting loop ---------
dl5:
        cmp     bp,0            ;test skip count
        je      ok_plot
        dec     bp
        jmp     short no_plot
ok_plot:
        call    plotdot         ;plot a dot
;
; Most of the following code is modified by the previous sequence.
; The 1111H's are dummy values that are always overlayed.
;
modify_base     label   byte
no_plot:
        inc     cx              ;INC/DEC CX/DX:  adjust long axis ptr
        add     bx,1111H        ;Xdist or Ydist: adjust duty master
        cmp     bx,1111H        ;Ydist or Xdist: check cycle position
        jle     dl6             ;skip if short axis is still ok

        sub     bx,1111H        ;Xdist or Ydist: adjust duty master
        inc     dx              ;INC/DEC DX/CX:  adjust short axis ptr
dl6:
        dec     di              ;di is used as counter
        jge     dl5             ;do next dot if not finished
                ;-----------------------------------------
        ret                     ;far return back to caller
drawline endp

;this routine plots the pixel at column CX  (0-319) or (0-639)
;                                 row    DX  (0-199)
;                                 color  AL  (0-3)   or (0-1) in high-res
; note: a much faster routine
;       can be written to plot dots
;
plotdot proc    near
        push    ax
        push    di              ;BIOS destroys these registers
        push    bp

        mov     ah,12           ;write_dot function
        int     10H             ;  video I/O call

        pop     bp
        pop     di
        pop     ax
        ret                     ;near return to DRAWLINE proc
plotdot endp
cseg    ends
```

```
        page ,132
        extrn drawline:far

;       Dan Rollins (213) 246-5021
;
;        TESTLINE----sample driver for DRAWLINE
;-----------------------------------------------
stack   segment  para stack 'stack'
        db       64 dup('stack   ')
stack   ends
;-----------------------------------------------
data    segment para public 'data'

parms   dw       7 dup (?)

; equates make it easier to program access to arguments

X1      equ      word ptr [si]
Y1      equ      word ptr [si+2]
X2      equ      word ptr [si+4]
Y2      equ      word ptr [si+6]
COLOR   equ      byte ptr [si+8]
LEN     equ      word ptr [si+10]
SKIP    equ      word ptr [si+12]
data    ends
;-----------------------------------------------
testseg segment para public 'code'
        assume  CS:testseg, DS:data

testline  proc  far
        push     ds             ;save program prefix segment for exit
        xor      ax,ax
        push     ax             ;word of 0000 for exit return address

        mov      ah,0           ;set mode function
        mov      al,4           ;med-res graphics
        int      10H            ;initialize graphics mode

        mov      ax,data
        mov      ds,ax                  ;set up data segment
        mov      si,offset ds:parms     ;point si to line parameter block

        mov      COLOR,3        ;white
        mov      X1,0           ;start is top left of screen
        mov      Y1,0
        mov      X2,319         ;end is bottom right
        mov      Y2,199

        mov      LEN,0          ;display entire line
        mov      SKIP,0
        call     drawline
        ret                     ;program exits to DOS (or DEBUG)
testline endp
testseg  ends
         end     testline
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dBASE II

by Frankie Detesto

At some point in your dBASE II career you will want to transfer a text file or wordprocessed file into a .DBF format so you can manipulate the information better. You may have a list of names and amounts that you have previously saved in a standard text file. If you have DOS 2.0 you are aware of the fact that you can send a sorted directory to a text file. You could then transfer this directory into dBASE II and calculate various file size totals.

Converting text files into .DBF format is quite easy to do. Let us say we have a text file called NEW.TXT that looks like this.

```
Bill Larson      35    Male    Carpenter
Vince Wilson     22    Male
Betty North      32    Female  Marketing
Josh Graves      19            Driver
Linda Evans      29    Female  Director
```

The fields must all be stored at the correct column for this transfer to go smoothly. It does NOT matter that they are of varying lengths or that certain fields are missing, just so every field is at the same column position. Our longest record is 51 columns. This will be the record length of our dBASE II file. We decide to make the LAST field a little longer just in case we miscalculated. So the final record will be 55 characters long.

We then CREATE a .DBF file called WORKER with this layout:

```
NAME,C,16
AGE,C,8
SEX,C,8
JOB,C,23
```

The rest of the commands are as follows:

```
USE WORKER
APPEND FROM NEW.TXT SDF
* check your work
LIST
USE
```

From this point you can chose to condense your data into smaller quantities, make the AGE into an actual number and compress the SEX into a smaller field. This might best be accomplished by creating another .DBF file and then transferring the fields to it while at the same time shrinking them.

Ideally your initial text file would look like this.

```
Bill Larson   35Male   Carpenter
Vince Wilson22Male
Betty North   32FemaleMarketing
Josh Graves   19       Driver
Linda Evans   29Female Director
```

No spaces between the fields.

dBASE AND BASIC

Time and time again I find myself going back to the dBASE II manual and looking up the STRING to NUMBER conversion expressions as well as dBASE equivalent for BASIC's STR$ and MID$ functions. It would have been nice if the commands were exactly the same but they are not. The hassle of it all led me to make a list of the most frequently used conversion commands. They are listed here for your convenience.

To extract a section of a STRING (in BASIC a MID$) you:

```
STORE $(A$,2,3) TO PIECE$
```

Take the variable A$, start at position 2 and take 3 characters, store the section in PIECE$.

The dBASE equivalent to INSTR looks like this:

```
STORE @("Z" ,A$) TO FOUND
```

You are trying to find a "Z" in the string A$. The result is 0 if it does not exit or a value if the letter is found.

BASIC's INSTR function is also represented another way:

```
DISPLAY FOR "Inc" $COMPANY
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This searches every COMPANY record in you data base and displays the record if "Inc" is in the field

One final NUMBER to STRING function. We may wish to index on a number field and the index is preferably a STRING and not a NUMBER.

```
STORE STR(8700,5,0) TO ITEMNUM$
```

The NUMBER 8700 is converted to a STRING plus a space and saved in ITEMNUM$.

POKE

An undocumented command in dBASE II is POKE. It is used just like BASIC's POKE. This gives you plenty of room to hang yourself if the urge strikes.

TWO NEW PUBLICATIONS

dBASE II users can now subscribe to two new magazines. One of them is published by Ashton-Tate for $18 a year. It is called dNEWS and you should have received the first two issues. If not then give them a call. After issue #3 you must subscribe.

The other magazine is called *The Data Based ADVISOR* and it is from Data Based Solutions. The subscription rate is $35 per year. For more information contact:

Data Based Solutions
ph. (800) 336-6060
1975 Fifth Ave. Suite 105
San Diego, CA 92101

□

The Norton Utilities™ POWER TOOLS™

Your data is in *danger!*

if you don't have the file recovery routines *UnErase* and *FileFix* just two of the incredibly useful *Norton Utilities*

File recovery:
UnErase - recovers erased files
FileFix - repairs damaged files
Diskette display:
DiskLook - complete disk displays and maps
Diskette patching and modifying:
SedMod - easy changes to diskettes
Hidden file control:
FileHide - interactive hidden file control
BatHide - automatic hidden file control
Time control:
Timemark - displays date, time, elapsed time
Screen control:
ScrAtr - sets DOS to work in any colors
Reverse - work in black on white
Clear — clears the screen for clarity
Diskette file sorting:
FileSort - keeps files by date or name
DiskOpt - speeds diskette access
Sound:
Beep - sounds a tone to alert you
File printing:
Print - prints files

IBM buys The Norton Utilities again and again. Shouldn't you buy a set?
Available from ComputerLand, other dealers, and directly from me for $80:
Peter Norton 2210 Wilshire Blvd. #186 H Santa Monica, CA 90403

"Norton's many utilities have received almost **universal praise**, and have become **an industry standard**. No home should be without them." —**New York PC Club**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Peeks and Pokes

PEEKS 'n POKES

There are a number of utility packages out for the IBM PC. What is most impressive about this utility package is its versatility. For the professional it gives you a set of source files and OBJ code that you can merge into your own programs without the need to spend hours developing it yourself. You also get a much needed series of subroutine modules to extract the information you repeated need about the diskette, monitor, keyboard and other system functions. The first time you use just ONE of these routines it will have paid for itself.

You can begin by running a MENU program that gives you the opportunity to run most of the utilities and to see what they do. You can examine the status of your system's keyboard, disk drive, video monitor, and printer as well as do direct READ/WRITE to disk. You will be able to find the disk directory at will and all the file attributes (size, date, time, hidden, etc.).

Another feature is the ability to determine where BASIC is loaded, it's start location, program offset, variable offset, and even the current line number of the program that is executing. When programs CHAIN to each other they use a common area in memory to store data, this is available also.

Last but not least you are shown "how to catch your own fish" rather than just being "given some fish" as it were. Section 14 in the documentation mentions how to discover the PEEKS and POKES on your own. It is so straight forward that you will kick yourself and say "now why didn't I think of that." What you do is this. Go into BASIC. Now BSAVE memory. Then change the attribute you are trying to locate in memory. Once again BSAVE memory to another file and use a file compare program to find the difference between the two files. The PEEKS utility supplies 3 ready to run BSAVE, COMPARE and CALC programs to do this for you.
PEEKS 'n POKES is a fine example of what each of us can aspire to. Supply a much needed piece of software, write a short and to the point manual, package it nicely, and give it a reasonable price for a change. What more could we hope for?

For Pascal users that do not have the advantage of directly peeking and poking with resident commands, you are supplied with the appropriate ASM and OBJ code to allow the calling of a provided assembly routine which does this for you.

It is rare to find a package that is both ideally suited for the novice programmer as well as the professional one. If you have developed some of this code on your own and have not been looking forward to writing the rest of it you are in luck.

The utilities come in your standard IBM style plastic binder with a 38 page manual explaining all you need to know about PEEKing, POKEing, XORing, XANDing and so on. This serves to explain how memory is altered with the supplied utilities.

Here are the files you receive:

```
57 File(s)

BEEP      COM        11
BSAVE     BAS       640
CALC      BAS      1024
CLS       COM        24
COLOR     COM        56
COM       BAS      2048
COMPARE   BAS      1920
COMSWAP   BAS       896
CONFIG    BAS      1536
CRT       BAS      2688
CRTDEMO   BAS      1152
CRTSWAP   BAS      1024
DATIME    COM       131
DISK      BAS      1408
DOS       BAS     17536
DOSC      BAS       512
DOSI      BAS      1536
DOSP      PAS      1147
DOSSUB    ASM      2026
DOSSUB    OBJ       172
DOSSUBI   ASM      2136
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
DOSSUBI   OBJ     160
INOUT     BAS     853
INSUB     ASM     554
INSUB     OBJ     128
KEYOFF    BAS     384
KEYPLUG   BAS     768
KEYREAD   BAS    1408
KEYSET    BAS    1408
LPT       BAS    1024
LPTSWAP   BAS     896
MENU      BAS    1792
MISC      BAS    1792
MONO      COM      51
OUTSUB    ASM     571
OUTSUB    OBJ     128
PAGE      COM       8
PEEKPOKE  BAS     758
PEEKSUB   ASM     859
PEEKSUB   OBJ     128
POKESUB   ASM     850
POKESUB   OBJ     128
READC     BAS    2432
READI     BAS    4096
READSUB   ASM     859
READSUB   OBJ     128
READSUBI  ASM    1197
READSUBI  OBJ     128
REVERSE   COM      24
SPACEC    BAS    1024
SPACEI    BAS    2688
SPACSUB   ASM    2011
SPACSUB   OBJ     212
SPACSUBI  ASM    2345
SPACSUBI  OBJ     228
UNP       BAS     256
WAIT      COM      69
```

Functions of some of the programs:

```
CONFIG    - System Configuration
UNP       - Unprotect BASIC programs
KEYREAD   - Read keyboard status
KEYSET    - Set keyboard
KEYPLUG   - Plug data into keyboard
KEYOFF    - Turn keyboard off/on
CRT       - Read video status
CRTSWAP   - Swap CRT displays
CRTDEMO   - Poke colors & characters
DISK      - Read disk status
LPT       - Read parallel printer
LPTSWAP   - Swap parallel printers
COM       - Read RS-232 adapter status
COMSWAP   - Swap RS-232 adapters
DOS       - DOS/BIOS function calls
READI     - Read file directory
SPACEI    - Space used/left on disk
BASIC     - Interpretive BASIC
MISC      - Miscellaneous functions
BSAVE     - BSAVE files for COMPARE
COMPARE   - Compare BSAVE files
CALC      - Calculate peeks and pokes
```

You may merge the code into your own programs and distribute them with no royalty fee what-so-ever. Also the DOS 2.0 version is already out. □

PEEKS 'n POKES by Brett Salter
80 column monitor required
price $30.00 plus $2.50 postage
DATA BASE DECISIONS
14 Bonnie Lane
Atlanta, GA 30328

(404) 256-3860

TELECOMMUNICATIONS TYPESETTING

COMPUTER OWNERS: If you have a modem, text editor, and communications program, you should consider the improved readability and cost savings provided by *TYPESETTING* your program documentation, lengthy article or manuscript, newsletter, or other text instead of just reproducing it from line printer or daisy-wheel output. Computer typesetting by telephone offers you high quality, space-saving copy that creates the professional image you want! Hundreds of type styles to choose from with 8 styles and 12 sizes "on line." And it's easy to encode your copy with our designated typesetting commands.

CONFIDENTIALITY GUARANTEED
— bonded for your protection —
PUBLICATION DESIGN, EDITING, & PRODUCTION

EDITING & DESIGN SERVICES, INC.
30 East 13th Avenue Eugene, Oregon 97401
Phone 503/683-2657

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ LIBRARY

C UPDATE

Here is a complete listing of all the books we know about concerning the C language. They all have their own place in your C library. We will briefly describe each one and put them in the order we would suggest purchasing them.

The C Programming Language by Kernighan & Ritchie

This is the book which defines the language, written by the authors of the same. Since this is the book to which all other texts and compilers refer, it is almost a prerequisite to obtain and read a copy of this book first. The content is excellent, both from a C point of view and from the ideas it imparts as regards GOOD programming. (Available from Southwater Corp.)

Learning to Program in C by Thomas Plum

If you had to choose only one textbook to learn the C language from, this book would be it. A thorough and complete tutorial manual. Every point that is made is illustrated with the appropriate code showing you how to integrate the idea into a working program. This sort of balance is difficult to achieve in a programming manual. Thomas Plum, the author, has taught C and UNIX seminars for years and this book obviously reflects his experience as an instructor as well as his ability to explain difficult concepts in programming.

The book assumes no previous experience with C, but moves rapidly enough so that someone with a C background will find many aspects of the book useful. C has many unique features that Thomas Plum emphasizes. You will learn to appreciate the aspects of C that make it a wonderful language. Other less complete programming texts leave you in the dark about how to access the special functions in another language. If BASIC was your first language and you picked up C on your own, you could continue to code merrily along still thinking within the constraints of BASIC. Every language has its areas of freedom, and C is such an open environment you might never fully enjoy the features it has unless a more experienced programmer points them out. Using a function in its most simplistic fashion is not too difficult, but putting together several functions in a creative way is the key to talented programming and getting the most out of the language you are using. *Learning to Program in C* addresses all of these points, showing you how to use fully the tools that C provides.

As with every language text that we find most valuable, this one covers the authors approach to Software Development; analysis, design, implementation, and maintenance. But if there is one subject that we need more than any other these days, which Thomas Plum continually stresses, it is how to program in a way that maximizes PORTABILITY. C may be the only great white hope we have left in terms of transportability.

The book is approximately 350 pages. Every chapter provides several programming problems for you to solve. The answers are provided in the corrosponding chapter appendix.

Plum Hall also sells a "C Programming Standards and Guidelines" manual for installations that need to define limits within which they can maintain a high degree of consistency and portability. (Supplied in both a U format for UNIX or a W format for Whitesmiths.)

Learning to Program in C
price $25 available from:
PLUM HALL
1 Spruce Ave.
Cardiff, NJ 08232
ph. (609) 927-3770

The C Puzzle Book by Alan Feuer

The title is a little misleading, the book is actually a series of small pieces of source code in C. You are asked to explain what each one does, what the output of a given routine will be, and so on. You might call it a laboratory manual for working with C or perhaps a test/workbook approach to understanding C functions.

For many of us who wish to learn C the only tools we have are the compiler we buy and the books we can find. The chances of finding another person in our neighborhood who worked for Bell Labs for 10 years is not too good. Self-test books like the *The C Puzzle Book* fill a much needed gap in this area. By using this workbook approach we can find our weak points in understanding the C language. One very nice aspect of the book is the fact that it loosely follows the chapters in Kernighan and Ritchies book. You can read a chapter in one and then test yourself in the other. Alan Feuer taught C programming at Bell Labs. (Available from Southwater Corp.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Notes by C.T. Zahn

You could best describe this monograph as an analytical look at the C programming language. It is a reference manual for the darker corners of the language. The book is very succinct and to the point. The syntax, capabilities, structures and program flow are all briefly examined and commented on. It is published by the Yourdon Press. If you are familiar with Edward Yourdon and his analysis concepts this book is very much of the same genre. C.T.Zahn takes us through a concise look at the features that C has to offer. Professional programming techniques are mentioned throughout its 100 pages. Another good perspective on the qualities that C has to offer and how to go about incorporating them. (Available form Southwater Corp.)

Other Books:

The C Primer by L. Hancock and M. Krieger

For a mid-range primer, between the thorough PLUM book and the "easy does it" Purdum book (which follows) you could choose this one by Hancock and Krieger. The two authors combined their talents. Hancock is the C expert and Krieger is the experienced writer. The result is a smooth reading introduction to C, no previous programming experience is necessary.

C Programming Guide by Jack Purdum

If you are a beginning BASIC programmer and have done few software projects, then this book is a possibility. It takes you from BASIC to a first beginning level of C and mentions both languages in the process. The book holds your hand as much as possible and tries to take you to first base. If you want to get to second base then you will need another book. A better title for the book might be "A Beginning C Guide for Novice BASIC Programmers." Many programming texts contain a great deal of information and are somewhat overbearing to novice programmers, it is not the case with this book. However if you are reasonably capable in another language then this textbook would probably not take you as far or as fast as you would want to go. (Available from Southwater Corp.)

If you are in the market for a C tutorial book, much of your decision should be based on your previous experience in programming and which style of books you prefer; do you learn by a careful step by step approach with many references to BASIC, or does you taste tend toward more code and the explanations that surround it.

SOUTHWATER CORP.

This company sells exclusively programming related textbooks. Their 8 page catalog includes books covering UNIX, C, Ada, APL, Pascal and Modula 2. If you have had difficulty acquiring computer language books from your local book store you should definitely get a copy of Southwater's free catalog. The book postage is pre-paid and you get increasing discounts with quantity orders. (Enclose payment with your order or phone for a credit card order.)

PROGRAMMING TEXTS

THE C PROGRAMMING LANGUAGE
by Kernighan and Ritchie
$21.50 postpaid

THE C PUZZLE BOOK
by Alan Feuer
$14.50 postpaid

C NOTES
by C.T. Zahn
$20.00 postpaid

C PROGRAMMING GUIDE
by Jack Purdum
$19.50 postpaid

THE C PRIMER
by Hancock and Krieger
$16.50

Enclose payment and send to:

SOUTHWATER CORP.
Dept. PJ
30 Mowry St.
Mt. Carmel, CT 06518
ph. (203) 288-0283

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C COMPILER UPDATE

There are several C compilers on the market. We have heard many good things about Computer Innovations CI-86 compiler. It is a full implementation of C with 8087 support etc.

Almost every C compiler is in a process of being "revised" and updated periodically. They are all in a state of continued improvement with an eye for portability.

MICROSOFT has announced a C compiler for the IBM PC running under PC-DOS. In actuality it is the Lattice version of the C compiler marketed by Lifeboat. Both Lattice and Lifeboat have agreed to allow Microsoft to market the C compiler with Microsoft's green packaging etc. Microsoft has added their own Linker and enhanced the compiler library with several routines. Otherwise it remains the Lattice version. The Microsoft version should be available this summer.

WHITESMITHS has announced a C compiler for the IBM PC. It should be ready to ship within a week or two after you get this issue of PJ. As one of the well respected companies supplying C compilers it will definitely be a prime contender for the C market.

Another product announced by Whitesmiths is a Pascal to C conversion program. If you are writing programs in Pascal and desire the portability that C offers then you should look into this product. We just talked with Whitesmiths before press time and will have more information in our next issue. (Or you can call Whitesmiths at (617) 369-8499.)

PROGRAMMERS/WRITERS

Your lengthy letters, comments, pieces of code and articles should be sent on diskettes. Preferably in standard ASC II files. WordStar and EasyWriter formats are also convenient. Illustrating your programming ideas through actual source code is by far the best way to clarify a point. Software is like Architecture, the more we see of each others work, the more unified our own designs become. Forward your ideas to PJ, 2765 Potter St., Eugene, OR 97405.

Now-chain Pascal programs and pass data between them with...

CHAIN™

Developed exclusively to improve Pascal programming efficiency on the IBM Personal Computer! This time-saving software package chains programs together to achieve the effect of increased RAM, *without* increasing your hardware! CHAIN saves you time and helps you develop more comprehensive menus. It's an absolute necessity for developing or expanding user-friendly programs.

10-Day Money-Back Guarantee!
See what CHAIN can do in your own programming applications. If not completely satisfied, return the diskette and manual within 10 days of receipt for a full 100% refund!

Order Today! Mail check or money order for $46.95, or call 1 (312) 298-3610

KEYSoftware Inc.

VISA MasterCard

Dept. 24 • 2350 E. Devon Ave. • Suite 138 • DesPlaines, IL 60018
Dealer Inquiries Invited

OPT-TECH SORT™
FORMERLY OPTIMUM SORT

SORT/MERGE program for the IBM-PC

— NEW Version 1.1 —
Contains all the most desirable features!

Features

- Sort ascending or descending on up to nine fields
- Ten input files may be sorted or merged at one time
- Written in assembly language for **high performance**
- Supports variable and fixed length records
- Supports all common data types
- Filesize limited only by your disk space
- Dynamically allocates memory and work files
- Can be run from keyboard or as a batch command
- Can be called as a BASIC subroutine
- Easy to use, includes on-line help feature
- Full documentation — sized like your PC manuals
- **$75** — VISA, M/C, Check, Money Order, COD, or PO
  Quantity discounts available

To order or to receive additional information write or call:

OPT-TECH DATA PROCESSING
P.O. Box 2167 Humble, Texas 77347
(713) 454-7428

Requires PC-DOS, 48K and One Disk Drive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOFTWARE HOUSES

SOFTSEL

Softsel Computers Products, Inc. is the largest distributor of personal computer software. They market more than 2,500 programs and books to over 3,500 computer retail store dealers. If a program you have written is accepted by them then you have an unquestioned advantage in terms of distribution channels. Exposure and contact with as many stores as possible increases your future sales. Much of their success is a reflection of their dealer support. Once again we find that "support" is the formula for corporate success in the computer world.

Softsel, in their "Product Submission Guidelines" clearly lay out for you the scope and process of their evaluations. They indicate that your submission will be "added to an evaluation database according to priority and date of receipt." Obvious they receive a fair amount of software to review. The turn around time is from 4 to 12 weeks.

Softsel will not sign any "non-disclosure" agreements. They also encourage you to pursue other avenues of distribution while they are considering your package.

If your software is accepted and you return their distribution agreement, then you should be ready to fill their orders. Softsel is normally a DISTRIBUTOR, not a company that buys the sole RIGHTS to your software and then pays you a royalty. The usual agreement is one where you become a vendor and fill their orders for your product. No commissions, consignments or royalties are paid. Very straight forward in terms of your self protection. And no doubt they order in large quantities. Also no "exclusive" distribution rights are required. You may market your product through anyone else you choose. If you are as big as Softsel, competition from other companies is of little concern.

Their "Product Submission Guidelines" includes 8 pages of questions that you need to answer. Most of it serves to clarify the first 30 questions an evaluator would ask. This is designed to speed the submission process; which computers will your software run on, what system configuration is required, describe the features of your package, what is the documentation like. Other general questions focus on printer compatability, operating system required, language the program is written in, etc.

The success of your submission venture with Softsel will depend heavily on how professional your product looks. Tasteful packaging and clean, type-set documentation should be a prerequisite. You might consider submitting your software to smaller companies first and see what response you get. If they offer suggestions for better packaging or recommend rewriting the manual then you are one step closer to a well integrated product. Take a look at other software marketed by Softsel and see what they like. Try and find a list of the packages they already carry and determine whether yours will fill a gap, being sure to mention this fact when you finally decide to send them your programs.

Softsel will inform you of their desire to market your product or not. They do not normally return your submission. If you want it back then you must include a written request to that effect.

For more information and a copy of their "Software Submission Guidelines" contact:

SOFTSEL
Software Submissions
546 North Oak
Inglewood, CA 90302
Ph. (213)-412-1700

C SCREEN EDITOR

CSE: A full-screen text editor written in C

$60 Includes complete source code

- Powerful command set includes cursor control, find/replace, block move, file inclusion, and nested macro commands.
- Full-screen editing, including cursor control, insert/overtype, single and multiple line insertion and deletion, and forward and reverse scrolling.
- Commands to perform large-scale operations on the text buffer, such as find, find and replace, move the text pointer, read input, and write output.
- A macro facility which allows grouping of a set of commands for repeated execution.
- Ability to work on files larger than the text buffer size.
- Complete configurability of control sequences, such as cursor keys and editor control strings, to allow customization of CSE for a specific terminal. CSE is configured to use special editing and function keys on the IBM PC.

NMD Northwest Microsystem Design

P.O. Box 10853 • Eugene, OR 97401 •

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE KNOW YOU'RE SERIOUS ABOUT THE IBM PC.

SO ARE WE!

WE KNOW WHAT YOU WANT.

- *Detailed how-to articles*
- *Programming tips*
- *In-Depth product reviews*
- *Problem reporting and solutions*
- *Code listings*

WE'VE ALWAYS HAD IT ...

Some features in previous issues were a technical look at the system unit, number crunching, utility programs, asynchronous communications, monitor and printer character sets, personalizing DOS, BASIC vs. Pascal, menu generation programs, unleashing your printer, evaluation of hard disk drives, and plotting data with PC graphics.

WE ALWAYS WILL.

Features for 1983 will include extensive information about networks, database managers, spreadsheets, operating systems, the 8087 chip, professional program development, and much more.

WE KNOW WHICH MAGAZINE YOU NEED ...

PERSONAL COMPUTER AGE

The Definitive Journal for the IBM Personal Computer User

CALL TOLL FREE: 800-824-7888 operator 77
California only: 800-852-7777 op. 77

I NEED WHAT YOU'VE GOT! **ENTER MY SUBSCRIPTION TODAY!**

☐ I WANT INFORMATION ABOUT BACK ISSUES, TOO!

☐ PAYMENT ENCLOSED ☐ BILL ME

CHARGE MY:

☐ VISA CARD NO. _____

☐ MASTERCARD EXPIRATION DATE _____

AUTHORIZED SIGNATURE _____

☐ 1 yr. (12 issues) $24.00
☐ 2 yrs. (24 issues) $45.00
☐ 3 yrs. (36 issues) $64.00
☐ NEW ☐ RENEWAL

FOREIGN SUBSCRIPTIONS
1 YR. (12 ISSUES)
CANADA/MEXICO $29.00

OTHER COUNTRIES,
WRITE FOR RATES
PAYMENTS MUST BE
IN U.S. FUNDS DRAWN
ON A U.S. BANK

NAME _____

ADDRESS _____

CITY _____

STATE _____ ZIP _____

Personal Computer Age

Department PJ 5
P.O. Box 70725
Pasadena,
California 91107

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PROGRAMMER'S JOURNAL

SUBSCRIPTION

Include your:

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

1 year bi-monthly subscription rate:
$24 U.S. ($30 Canada, $40 other)
☐ Check enclosed
☐ Bill me

Signature _____
Begin my subscription with: ☐ First Issue ☐ Current Issue

Your system configuration:

Memory (K) 64 128 256+

Disk Drives 1 2 160 320 HD

Monitor: Color Mono

Languages you use: _____

Languages interested in: _____

_____

Send to:
PROGRAMMER'S JOURNAL
2765 Potter St.
Eugene, OR 97405
Ph. (503) 687-0398

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Next time. . . ASM, BASIC, PASCAL, C

$4.00

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED